8-1-1995

Design and Implementation of a Fuzzy Logic Speed Controller for an Internal Combustion Engine

Craig Jensen

DESIGN AND IMPLEMENTATION OF A FUZZY LOGIC SPEED CONTROLLER FOR AN INTERNAL COMBUSTION ENGINE

by

Craig Jensen
Bachelor of Science, University of North Dakota, 1993

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota
August
1995

This thesis, submitted by Craig Jensen in partial fulfillment of the requirements for the degree of Master of Science from the University of North Dakota, has been read by the Faculty advisory committee under whom the work has been done and is hereby approved.

N. N. Bengiamin
(Chairperson)

J. Hoolman

J. Wang /by N. Beng

This thesis meets the standards for appearance, conforms to the style and format requirements of the Graduate School of the University of North Dakota, and is hereby approved.

Harvey Knull
Dean of the Graduate School
July 24, 1995

PERMISSION

Title Design and Implementation of a Fuzzy Logic Speed Controller for an Internal Combustion Engine

Department Electrical Engineering

Degree Master of Science

In presenting this thesis in partial fulfiliment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the chairperson of the department or the dean of the Graduate School. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Signature Craig Jensen

Date 7-24-95

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figures Page

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my research advisor, Dr. Nagy N. Bengiamin for his help, support, and advise throughout this thesis. I would also like to thank the members of my thesis committee, Dr. Joallan Hootman and Dr. Jun Wang, for their review and comments on my thesis, and Brad Trostad for his expert advise on C++ programming.

ABSTRACT

Internal combustion engines are challenging to model and control. Uncertainties and nonlinearities pose operating problems for classical controllers. Delays inherent to the engine combustion cycle tend to introduce overshoot and oscillations in most control schemes. Most work that has been done in dealing with delays requires the designer to have extensive knowledge of the system to be controlled. Engines are very difficult to model accurately, thereby ruling out most of these techniques. Fuzzy logic is well suited to this problem, since an accurate model is not needed for design, and it is known to be robust to nonlinearities and parameter variations.

The objective of this thesis was to design and implement a fuzzy logic controller to control the speed of a Honda EM3500S portable generator. This new fuzzy controller maintains the robustness of traditional fuzzy logic to nonlinearities and it is also more robust to delays. The control scheme uses dual fuzzy logic control modules in parallel. One of the modules is a traditional fuzzy scheme and the other is a simple two membership fuzzy scheme tuned to reduce oscillations. For optimal performance this second module requires dynamic adjustment of parameters such as input and output gains in response to the system's current operating condition. The result is a control scheme that offers reduced overshoot and oscillations.

The new control scheme was compared to the classical PID and the traditional fuzzy logic controllers. These comparisons were done via computer simulations and laboratory implementation and testing. A windows based C++ program was developed to realize and test the new controller. The better performance of the new control scheme was illustrated.

1. INTRODUCTION

Internal combustion (IC) engines are one of the most widely used sources of power by both industry and consumer. In industry, trucks and heavy machinery are commonly driven by IC engines. Some consumer uses are in automobiles, lawnmowers and portable generators. Most engines need some type of feedback control system to keep them operating within their design specifications. Several examples are cruise control and emissions control in automobiles and speed control for a portable generator. This chapter is devoted to the fundamentals of engine control, its applications and some of the work done in the past in this area.

1.1 Engine Control

Engine feedback control systems are designed to sense operating conditions and respond with a control signal to adjust an engine input. One example is portable and emergency backup generators. Portable generators are used in places were electric power is needed but no direct connection to an electric utility company is possible. Some examples are a lake cabin, a construction site, on an RV or as an emergency backup when the utility is out of service. Portable generators usually generate electricity at 120/240 volts and 60 Hz. This is the standard for consumer electricity, i.e. house outlet.

Some applications like electric heaters and lighting are tolerant of variations in electric power quality including voltage and frequency. In other equipment like electronics, microwaves, computers and television sets, it is very important to maintain high electric quality with tolerance as close to 120 volts and 60 Hz as possible. The industrial standards call for steady state power quality of ± 10V and ± 1 Hz. Sophisticated equipment may not operate or may even be damaged if the electricity deviates too much from its nominal value.

System voltage is controlled by the exciter coil of the generator. Exciting this coil creates a magnetic field in the air gap of the generator. The output voltage at the terminals of the generator is proportional to the strength of this magnetic field. The output voltage is also weakly dependent on engine speed. Since the engine speed will be kept constant via a speed controller this coupling is usually neglected. Therefore, output voltage is controlled by sensing the system voltage and adjusting the exciter current accordingly.

The speed of the IC engine is directly proportional to the frequency of the electricity it generates. For this reason it is very important to keep the portable generator running at its nominal speed. This is done by measuring the speed of the engine shaft or frequency of the electric voltage and adjusting the throttle position accordingly. Any time the electrical load changes the input-output energy balance is instantaneously offset. The kinetic energy of the rotating parts of the engine-generator adjusts itself instantaneously to maintain that balance. Since the kinetic energy is directly proportional to the square of the speed of rotating components, it is obvious that this speed is most

vounerable to this input-output balance. The purpose of the engine control system is to minimize kinetic energy variations by fast manipulation of input energy. The controller must adjust the throttle very rapidly to maintain tight speed tolerance. Figure 1 illustrates this system.

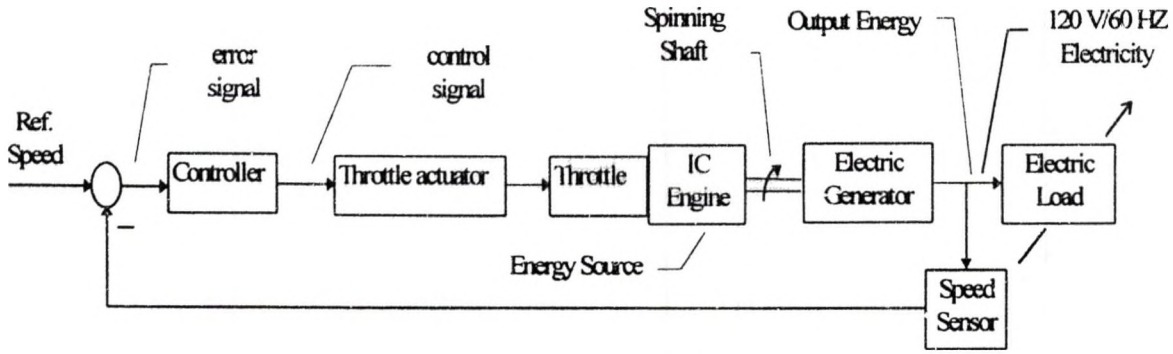

Figure 1. Engine-Generator Control System

Another example is automobile cruise control where the variable being measured is the vehicle speed and the controlled input is the throttle position. The system works by repeatedly measuring speed, comparing it to the set speed and adjusting the throttle accordingly. Disturbances such as hilly roads, wind and other environmental conditions cause the system to continuously adjust the throttle position to maintain the desired vehicle speed.

Many modern engines are equipped with several onboard control systems to handle the level of complexity associated with engine operation. Emissions and fuel consumption are a few of the systems controlled on modern engines. For obvious reasons, most of the advanced work in engine control is done by the automotive industry. A good overview of modern automobile engine control systems is given in [1-2]. Precise

control of the air-to-fuel ratio is the key to both fuel efficiency and meeting environmental emissions policy. Reducing the air-to-fuel ratio tends to lean the combustion process and thus reduces emissions and increases fuel efficiency. On the other hand it also tends to deteriorate the system's performance and thus reduces it's driveability. Development of air-to-fuel ratio control systems was addressed by Ishii, et. al., [3], among many other researchers. The main challenge in developing air-to-fuel ratio control schemes is the inherent nonlinear nature of the system compounded with the unavailability of needed highly accurate sensors. Engine combustion conditions vary due to changes in operating conditions such as temperature and altitude, and changes due to time such as deterioration of engine components. Many of the systems in use today are adaptive in nature to better deal with these uncertainties [23].

Several engine speed control schemes have been developed and reported in the literature [4-6]. Engine speed control systems can be either pure mechanical or a hybrid of mechanical and electronic components. In a purely mechanical controller, a flywheel is mounted on the spinning shaft of the engine to sense speed variations. The centrifugal force of the flywheel is linked to the throttle such that when the engine speed changes, the throttle is properly adjusted. Simplicity and practicality are the appealing features of this system. The disadvantages are, 1) each controller must be custom designed for its application and can not easily be used on another engine and 2) the controller must be tuned to run at a specific speed and cannot be easily adapted to a different operating speed. Advanced electronic based controllers require additional overhead such as more elaborate speed sensors and throttle actuators in addition to onboard electronics. The

need for these additional components is offset by the system being much more flexible to changes in design and the ability to implement more advanced digital control schemes.

Electronic controllers can be either analog or digital based. Digital based controllers usually require converting some sort of analog signal into a digital number so it can be manipulated by a microprocessor. The A/D conversion process and any required processing of the signal introduces a delay which tends to adversely affect the performance of the system. Analog controllers do not suffer from microprocessor delays associated with digital algorithms but are generally less powerful in features and more difficult to modify.

Development of electronic controllers usually begins with the creation of a mathematical model of the system to be controlled. Once an accurate model has been developed, the performance of the final system can easily be tested via dynamic simulation. Several CAD tools are available for dynamic simulation; MATLAB and TUTSIM are a few of the popular ones.

Numerous work has been done in the area of engine modeling [7-9,25]. A large portion of this work was spurred by the automotive industry in search of accurate engine models for the development of sophisticated controllers. Engines exhibit inherently nonlinear characteristics under different operating conditions. Internal combustion engines are also characterized by a time delay associated with the combustion cycle of the engine. Delays in the engine model tend to degrade the performance of controllers by introducing overshoot and oscillations. Most of the current research is in the area of modeling fuel consumption and emissions. These models have control inputs of air to

fuel ratio, exhaust gas recirculation, and spark advance, and outputs of fuel consumption, and emissions [8,25].

For speed control, a simplified model having throttle position as its only input and load torque and engine speed as the outputs is sufficient. Also, since the speed of the generator is usually kept within tight tolerance, much of the nonlinearities due to different operating speeds can be eliminated.

After the engine model is developed, it is used to design and test the controller on a computer simulator. This allows testing of different control schemes without possible damage to the actual engine. Once the design is verified via computer simulation it can be implemented with relative confidence.

1.2 Control Schemes

One of the classical electronic based controllers is the Proportional Integral Derivative (PID). PID controllers are used heavily for engine control because they are easy to design and implement. One drawback of the PID controller is that it is not known to be robust to changes in engine parameters. Robust systems maintain their performance despite changes in plant parameters. Tuning a PID controller involves simply adjusting it's gains. The proportional gain tends to speed up the system but in turn introduces oscillations. Integral gain tends to reduce steady state error but also introduces oscillations. The derivative gain reduces oscillations but it may slow down the system. The final design for a PID controller provides a compromise between system response speed and oscillations. PID controllers can be implemented either as analog or digital.

Analog PID controllers are realized using simple operational amplifier circuitry. Digital PIDs are microprocessor based where a set of equations are solved to determine the control signal.

Another class of digital controller is the fuzzy logic controller [12-14]. Fuzzy logic control has been used in many different areas including engine control [4-5], electric machine control [21], and transportation [22]. Fuzzy controllers are known to be robust to changes in plant parameters and nonlinearities and thus are of interest for engine control.

The basis of fuzzy logic is fuzzy set theory. In conventional crisp set theory, an element exclusively belongs to a single universe of discourse. In fuzzy set theory, elements can be partial members of sets. Fuzzy membership functions can take any one of several shapes. The most common shapes are the triangle and the bell curve. Triangular membership functions are preferred for control applications due to their associated simplicity. Bell curves are sometimes used in cases where continuous derivatives are needed for mathematical proofs. The first step in the fuzzy logic controller is the fuzzification of the inputs. In this process each input is assigned a value corresponding to its degree of membership to each input membership function. After the fuzzification process the fuzzified inputs are combined through a set of inference rules. These rules are selected based on a preexisting knowledge of how the system should respond under different operating conditions. Each rule consists of two parts, a condition and a conclusion. The conclusion determines what the output will be if the rule's conditions are met. After the inference process, the conclusions of all rules must be

combined to form a crisp output. This step is called defuzzification and can be implemented in several different ways. The easiest and most common for control applications involves summing the output of each rule that fired and dividing it by the sum of the inputs to each rule.

Tuning a fuzzy controller is usually done by adjusting the membership functions and the rules until the system performance criteria is satisfied. Adaptive techniques to automatically tune the parameters of the fuzzy controller have been developed [14-15].

1.3 Objective

The objective of this thesis is to design and implement a fuzzy logic controller that is not only more robust to engine nonlinearities and parameter variations, but also more robust to time delays than other controllers. A standard two input, five membership fuzzy controller will be designed and tuned to control the voltage frequency of a Honda engine generator system. This controller will then be modified by adding a second fuzzy controller in parallel with the first. The objective of the second controller will be to anticipate when overshoot will occur and inject a strong control signal to reduce both overshoot and oscillations caused by the system delay. This scheme will not affect the speed or robustness of the original controller but will counteract some of the degrading properties due to the system's time delays.

2. ENGINE MODEL

In this chapter a reference model for the Honda EM3500S portable generator will be developed. This model will be used to develop and tune the fuzzy controller developed in the following chapters. The EM3500S is rated at 3.5 kVA maximum and 3.0 kVA nominal. Electricity is generated at 60 Hz, 120/240 volts single phase at an engine speed of 3600 rpm.

Most of the initial work in engine control modeling was done by the automotive industry in its attempt to reduce both exhaust emissions and fuel usage. These models attempt to model both the engine throttle torque characteristics and also the emissions characteristics; see Cassidy et. al [8]. These models are complex and their parameter identification is challenging. For the purpose of speed control, a limited model for the throttle torque characteristics is usually sufficient. A simple model and parameter identification technique was developed by Morris, et. al.[9]. Min [20] used a similar model and identification technique that will be the basis for this chapter. Due to limitations of available hardware the parameter identification procedure outlined by Min was modified slightly. The throttle torque characteristics of the developed model are not affected but certain physically measurable engine variables, namely intake manifold pressure, are not determined.

2.1 Engine Model Description

The identification technique breaks the engine model into components and finds the transfer function for each one individually. It then combines the identified components to form the final model. The model is constructed mainly of linear components. This linearization makes the model valid only around a specific operating point. This operating point limitation does not pose a problem for the generator system because it will always be operated at a constant speed of 3600 rpm. The model assumes zero exhaust gas recirculation, constant spark advance and constant air to fuel ratio. Three system components were identified. They are, 1) throttle body, 2) intake manifold, and 3) rotational parts. The throttle body models the flow of the gaseous fuel air mixture through the carburetor. The intake manifold section models the flow of the gaseous mixture through the manifold and into the combustion chamber. A time delay exists between the throttle dynamics and the power output. This delay is inherent to the Otto cycle of the engine. The third component, (rotational parts), includes the inertia and friction of both the engine and the alternator together.

The variables of the engine obey the following functions

$$
\begin{aligned}
\dot{M}_I &= f_1(\Theta(t)) \\
\dot{M}_O(t) &= f_2(M_M(t), N(t)) \\
M_M(t) &= \int_0^t (\dot{M}_I(s) - \dot{M}_O(s))ds + M_M(0) \\
T_I(t) &= f_4(\dot{M}_O(t - t_D)) \\
T_F(t) &= f_5(N(t)) \\
N(t) &= \frac{1}{J}\int_0^t (T_I(s) - T_F(s) - T_L(s))ds + N(0)
\end{aligned}
\tag{1}
$$

where

$\dot{M}_I(t)$: rate of air mass entering the intake manifold
$\dot{M}_O(t)$: rate of air mass exiting the intake manifold
$M_M(t)$: air mass in the intake manifold
$\Theta(t)$: throttle angle (degrees)
$N(t)$: engine revolutions per minute (RPM)
t_D : delay time (s)
$T_I(t)$: produced indicated torque (ft.lb)
$T_F(t)$: total frictional torque loss (ft.lb)
$T_L(t)$: external load torque (ft.lb)
J : total inertia

Assuming that at time t=0 the engine is in its nominal mode of operation and therefore the variables derivatives, (initial conditions), are zero, the above relationships can be written in the linearized form,

$$\begin{aligned}
\Delta\dot{M}_I(t) &= \Delta\Theta(t) \\
\Delta\dot{M}_O(t) &= k_2\Delta M_M(t) + k_3\Delta N(t) \\
\Delta M_M(t) &= \int_0^t (\Delta\dot{M}_I(s) - \Delta\dot{M}_O(s))ds \\
\Delta T_I(t) &= k_4\Delta M_M(t - t_D) \\
\Delta T_F(t) &= k_6\Delta N(t) \\
\Delta N(t) &= \frac{1}{J}\int_0^t (\Delta T_I(s) - \Delta T_F(s) - \Delta T_L(s))ds
\end{aligned} \tag{2}$$

where Δ is used to indicate deviation from nominal values.

Since the original model is highly nonlinear, the relationships of (2) are only valid around a specific operating point. This will not be a problem with the engine generator because it will always be operated at nominal speed. If operated at speeds other than nominal, (3600 rpm), a new set of parameters would need to be found for each operating

speed. Typically 10 - 20 different sets of parameters are sufficient to cover all possible operating speeds [20]. The linearized engine model is shown in Figure 2.

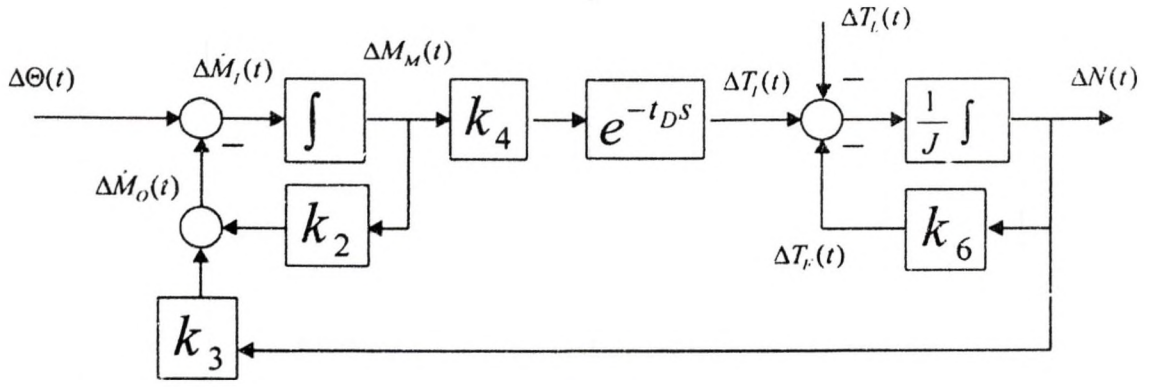

Figure 2 Linear Engine Model

2.2 Parameter Identification and Testing

Several experimental tests must be run to determine each of the parameters listed above. Each of the tests was run several times and average values were used for parameter identification. After a description of each test, a sample calculation will be given showing how the parameters were determined.

Static Test

The Honda engine was run at different combinations of speeds, load levels and throttle settings. For each operating condition the load in watts, throttle position in percent open and speed in rpm were recorded. The engine was run at operating speeds other than the nominal for the dynamic test which will be explained later. The data is given in Table 1.

Number	Throttle	Speed (rpm)	Power (Watts)	Torque (ft-lb)
1	15.6	3600	0	0.00
2	15.6	3300	193	0.41
3	15.1	2350	710	2.13
4	15.1	3450	0	0.00
5	22	3600	767	1.50
6	24.2	2900	1450	3.52
7	29	3600	1500	2.93

Table 1 Static Test Results

It was noted that a throttle opening of 15.6 degrees was needed to attain 3600 rpm at no load. This will be considered the nominal operating position, (zero displacement, i.e. $\Delta\Theta = 0$), for the throttle. The resulting block diagram is shown in Figure 3. k_4 was determined from entries 1 and 5 in Table 1 as follows,

$$k_4 = \frac{\Delta T_L}{\Delta\Theta} = \frac{1.50}{6.4} = .23 \tag{3}$$

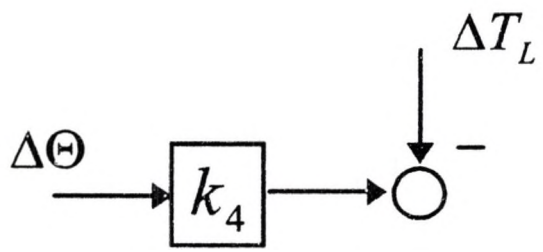

Figure 3. Block Diagram for Static Test

Constant Load Static Test

The engine is operated at two different combinations of load and speed. Since the results are recorded after the engine reaches a steady state operating condition, the block diagram reduces to the one shown in Figure 4.

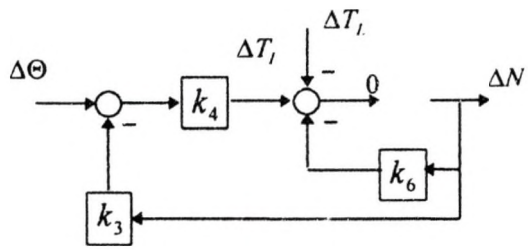

Figure 4. Block Diagram for Constant Load Static Test

From Figure 4,

$$\Delta T_I = k_4(\Delta\Theta - k_3\Delta N) \tag{4}$$

and $$\Delta T_I = k_6\Delta N + \Delta T_L \tag{5}$$

therefore, $$k_6\Delta N + \Delta T_L = k_4(\Delta\Theta - k_3\Delta N) \tag{6}$$

Using entries 1 and 2 in Table 1 and equation (6), the relationship between k_6 and k_3 is determined, eqn (7).

$$k_3 = -.23k_6 + .00126 \tag{7}$$

Run Down Test

In this test the engine is run at a specified speed then the ignition is turned off and the rotational dynamics bring the engine to a rest. A relationship between k_6 and J can

be obtained. In this configuration the block diagram of Figure 2 reduces to that shown in Figure 5.

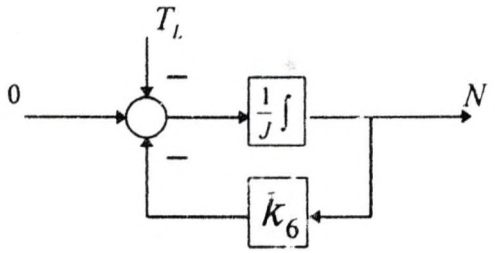

Figure 5. Block Diagram for Run Down Test

From Figure 5,

$$-T_L - k_6 N = J\frac{dN}{dt} \tag{8}$$

Since the load is zero,

$$\frac{dN}{dt} = -(\frac{k_6}{J})N \tag{9}$$

Therefore, using (8) and (9)

$$N(t) = N(0)e^{-\frac{t}{\tau}} \tag{10}$$

where, $$\tau = \frac{J}{k_6} \tag{11}$$

and $N(0)$ is the initial speed

This test was run with the initial engine speed at 4000 rpm. It was found that the engine slows to 2400 rpm in 1.49 seconds, see Figure 6. Solving equation (10) for τ yields $\tau = 2.92$ s. The relationship between k_6 and J, from (11), is as follows,

$$J = 2.92k_6 . \tag{12}$$

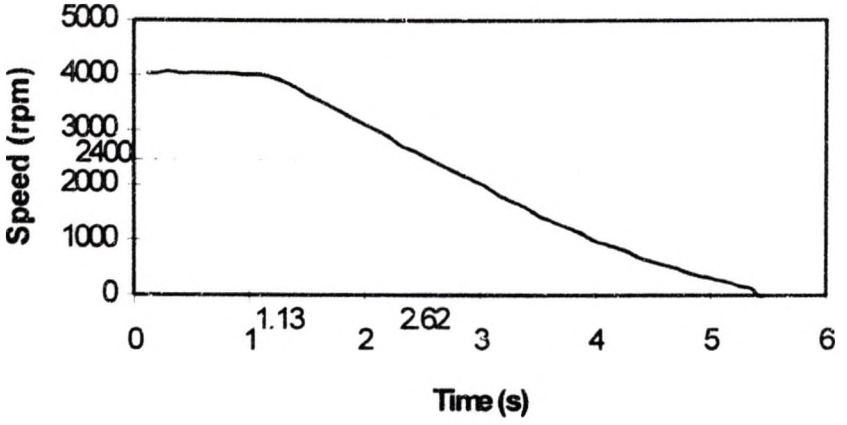

Figure 6. Run Down Test Results

Dynamic Simulation Test

This test is performed to determine the value of the manifold filling time constant, k_2, and estimate k_6 and also to verify the entire block diagram. The dynamic response of the engine is recorded for several different throttle/loading conditions. The engine parameters (k_2, k_3, k_6, J) are then found by trial and error via computer simulation of the dynamic model. Trial and error had to be used because of the lack of hardware to perform certain tests to identify the intake manifold dynamics and the rotational dynamics. A time delay is added to the dynamic model to simulate the delay between change in manifold pressure and change in output torque due to the Otto cycle. The delay could be anywhere between 1 and 2 engine revolutions. The minimum delay occurs when the throttle command is issued at the instant the intake valve opens after which the

engine must go through the intake and compression cycles before the first combustion process. The maximum delay occurs when the throttle command is issued at the instant the intake valve closes after which the engine must go through all four cycles before the next combustion. The average value for this delay is 1.5 engine revolutions. The average time delay is therefore found from equation (13).

$$t_D = \frac{(60)(1.5)}{N_{\text{nominal}}} \tag{13}$$

where N_{nominal} is the rated speed of the engine. The average delay is found to be 25.0 ms since $N_{\text{nominal}} = 3600$ rpm. Trial and error was then used to determine k_2, k_3, k_6 and J. The values were then fine tuned to match lab results. Table 2 shows the theoretically determined parameter values and the final values after fine tuning.

Parameter	Final Values	Equations (7) and (10)
k_2	.1	
k_3	.001018	.001007
k_4	.23	
k_6	.0011	
J	.00286	.003212

Table 2. Parameter Values

TUTSIM [24] was used to simulate the engine dynamics. The block diagram used is shown in Figure 7.

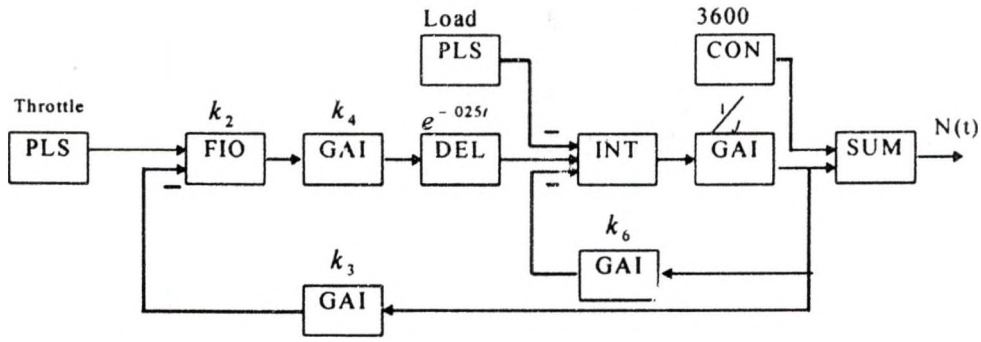

Figure 7. TUTSIM Block Diagram

2.3 Model Validation

The dynamic response of the engine was compared to actual lab runs for several different throttle/loading conditions. The results are shown in Figures 8 and 9. In Figure 8, the dynamic response of the engine is compared to that of the model in response to a step input of load (1.5 ft.lb) at 2 seconds and a throttle opening of 6 degrees at 11 seconds. Figure 9 shows a comparison of engine response to repeated application of 1.5 ft.lb of load with the throttle held constant at 15.6 degrees. Good agreement is clear in both cases.

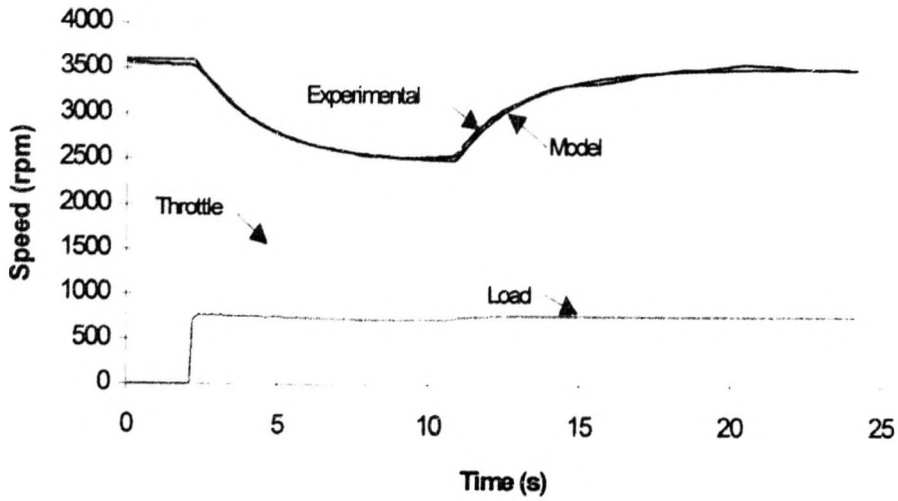

Figure 8. Load and Throttle Change Test

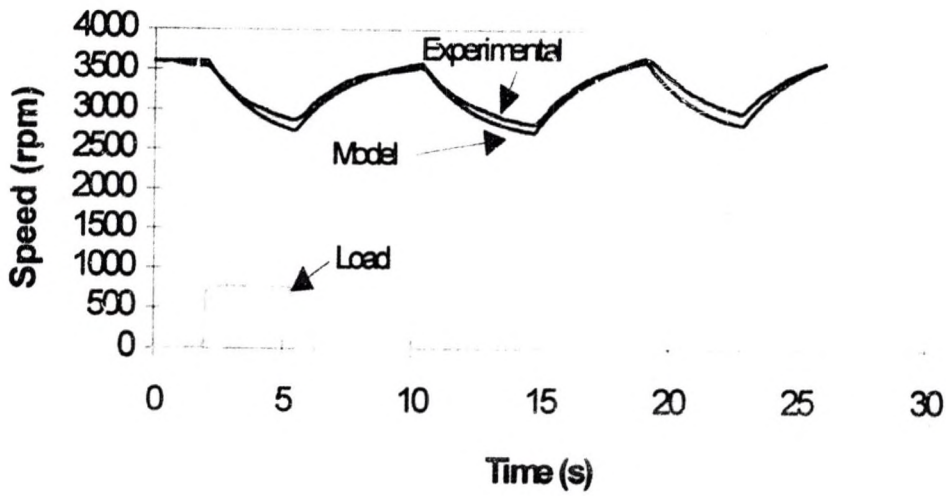

Figure 9. Repetitive Load Change Test

3. FUZZY LOGIC CONTROL SCHEME

In this chapter a fuzzy logic control scheme will be developed to control the speed of the Honda EM3500S portable generator. This controller will be tuned to deal with the problems associated with the delay that is inherent to the engine combustion cycle and also any microprocessor delay introduced by implementing a digital control scheme. A Windows based C++ program was developed to implement the new controller, a traditional fuzzy controller and a classical PID controller. The software will also simulate the dynamics of the Honda engine for full evaluation capabilities.

3.1 System Description

The original speed controller for the Honda engine generator is a purely mechanical system with a flywheel linkage to the throttle. This system works well at maintaining a static speed of 3600 rpm (60 Hz). One principle drawback of this system is the inability to operate at speeds other than 3600 rpm. Most applications in the United States require exclusively 60 Hz power but if the generator were to be used in Europe where 50 Hz power is the standard, major design changes would be necessary. An electronic control scheme offers the flexibility of easily changing operating speeds. Realization of electronic control schemes requires the addition of several components including a throttle actuator, a shaft speed sensor and signal processing electronics.

The first step in developing the new system was to remove the existing mechanical control system. This involved simply removing the linkage that connected the flywheel to the throttle. A high resolution stepping motor was then mounted as the throttle actuator. A stepper motor was chosen because of its ability to hold position tightly in the presence of vibrations. This stepping motor is equipped with its own microstepping controller and driver interface allowing a resolution of 2000 steps per revolution. The stepper controller is configured to accept an 8 bit digital control signal. The first 7 bits control the speed of the stepper and the 8th bit controls the direction.

The stepper controller can be configured to either control the position of the throttle or its speed and direction. Since throttle position is directly proportional to the torque output of the engine, throttle position control is the most direct way to control engine speed. However, implementing a controller with throttle position as the controlled variable requires the addition of an integrator to remove steady state error inaccuracy. Basic fuzzy logic does not have an integrator effect therefore it usually suffers from steady state error. Steady state error can be minimized by increasing the system gain but this can lead to chattering and system instability. Controlling the throttle speed (rate of change of throttle position) is equivalent to adding an integrator and thus eliminates steady state errors. It must be noted that adding an integrator to the system retards the phase of the system by 90 degrees, thus reducing its gain and phase margins which results in a less stable system. This implementation was chosen because of the required tight speed tolerance in operating engine generators.

An electronic frequency transducer was installed to indirectly measure the shaft speed. This transducer is connected across the terminals of the generator and it measures the frequency of the electricity generated. The output of the transducer is a 0-5 volt DC voltage corresponding to the measured frequency.

A 486/33 MHz based personal computer with A/D and digital output boards was chosen to implement the controller. The system works by repeatedly reading the shaft speed, determining the proper control action and sending out the control signal to the stepper motor. Windows based software was written in the C++ language to implement this system

3.2 Basic Controller

A traditional fuzzy logic controller was developed to control the speed of the Honda engine. The software implementation of the fuzzy controller was taken directly from Welstead [14]. It includes the three classical fuzzy control steps, namely fuzzification, inference and defuzzification. This software is flexible in that it allows the user to choose the number of inputs, the number and shape of the membership functions and the number of output membership functions. A simplified formula (14) is used to determine the crisp output control signal.

$$output = \frac{\sum_i (w_i z_i)}{\sum_i w_i} \tag{14}$$

where, w_i = output from rule i,

and z_i = gain for rule i

This simplification does not result in any loss of accuracy but it aids in improving the efficiency of the control code.

The inputs to the controller are e and e', where e is the difference between the engine speed and the desired speed and e' is the rate of change of e. Five triangular shaped input membership functions were chosen for both e and e', Figure 10. The membership functions are denoted as NL (negative large), NS (negative small), ZR (zero), PS (positive small) and PL (positive large).

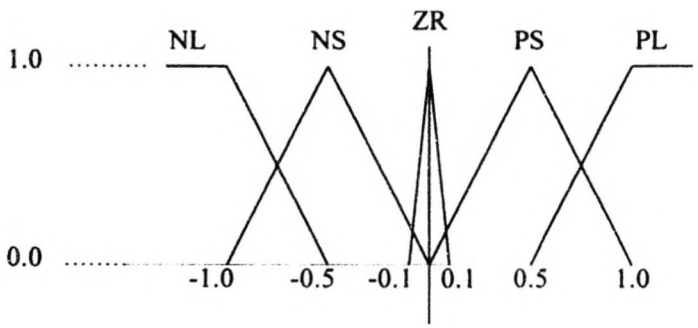

Figure 10. Input Membership Functions

The next step was to determine a set of inference rules to control the way the system behaves in response to the engine's state. A total of nine different crisp output values are used in the rule table namely; NL (-1.0), NM (-0.55), NS (-0.3), NVS (-0.1), ZR (0.0), PVS (0.1), PS (0.3), PM (0.55), PL (1.0). The inference rules are given in Table 3.

e \ e'	NL	NS	ZR	PS	PL
NL	NL	NM	NS	NVS	ZR
NS	NM	NS	NVS	PVS	PS
ZR	NS	NVS	ZR	PVS	PS
PS	NS	NVS	PVS	PS	PM
PL	ZR	PVS	PS	PM	PL

Table 3. Inference Rules

Since the input and output membership functions are normalized gains must be used to properly weight each of the inputs and the output. The error input gain was selected to be 0.003 which allows for a maximum input of 333.3 rpm before saturation. The derivative input gain was fine tuned to 0.001 via dynamic simulation. The output gain was set to 220.0 to match the response speed of the PID controller. Figure 11 shows a block diagram of the basic fuzzy controller.

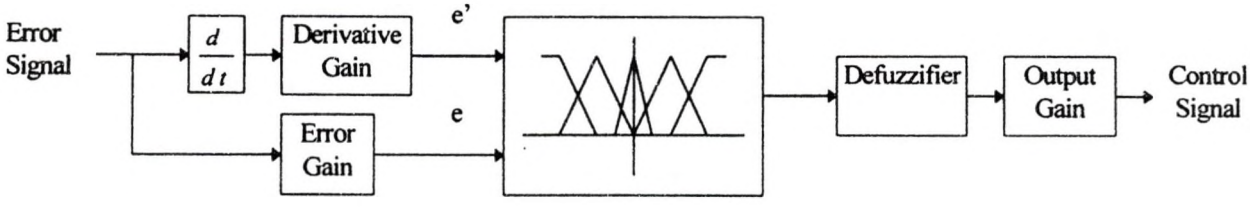

Figure 11. Basic Fuzzy Controller

Several source files are needed to compile and run this controller. The following is a list of the necessary files and a brief description of their function.

ENGINE.H

This header file contains definitions of the inputs, input memberships and outputs. It also has definitions for the functions contained in ENGINIT.CPP.

ENGINIT.CPP

This file contains the initialization code for the fuzzy controller. It was adapted from the example file FLBINIT.C. It has definitions for the entire fuzzy controller including inputs, membership functions and output rules.

FLFILES.CPP

This file contains code to read and write fuzzy systems to a disk file. This file was not modified.

FLOGIC.H

This header file contains the structures for the membership functions, fuzzy sets and initialization functions. It also has definitions for the maximum number of inputs, membership functions and outputs.

FLPRCS.CPP

This file contains the definitions for the fuzzy logic procedures and functions. No modification to this file is necessary for implementation.

3.3 The New Controller

Conventional fuzzy logic controllers are less susceptible to some of the problems inherent to classical PID schemes, namely overshoot and oscillations. In the presence of delays however, some of these problems can still occur in fuzzy logic control. Delays tend to retard the phase of a system and consequently reduce the gain margin resulting in a less stable system.

Several methods exist for dealing with delays. The first is to simply reduce the gain of the controller until desired stability conditions are met. This approach degrades the dynamic performance as it slows the overall system. Other more advanced methods have been developed including the Smith Predictor method described in [19]. This approach effectively deals with the problem of delays but it requires an accurate model of the system being controlled.

In an effort to illustrate the effect of delays on the performance of the engine, MATLAB was used to calculate Bode plots and determine the system gain margin. Gain margin is the maximum gain that can be added to the system before instability occurs. Figure 12 shows a block diagram of the closed loop engine control system containing the controller, the throttle actuator and the Honda engine-generator. The controller is a Proportional Derivative (PD) controller that is used later in this chapter. The throttle actuator is modeled as an integrator because the control signal governs the speed of the throttle. Block reduction techniques of [10] were used to reduce the engine model of Figure 2 to its transfer function given in Figure 12.

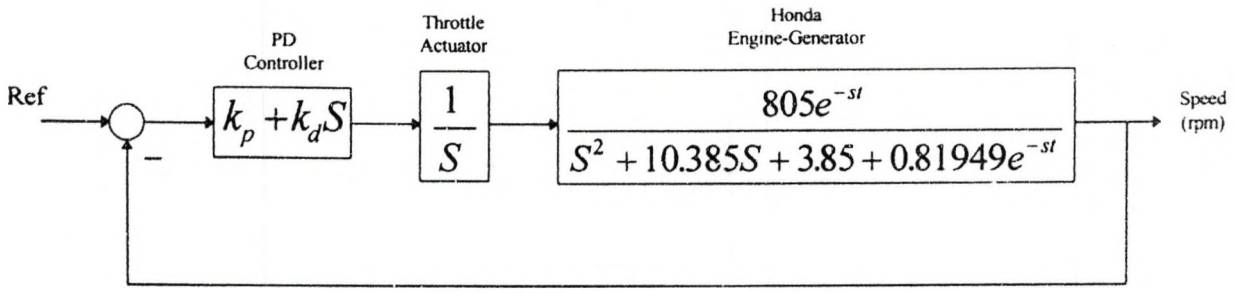

Figure 12 Closed Loop Control System

The open loop system transfer function with the gains (k_p and k_d) used later in this chapter is given as equation (15).

$$GH = \frac{805(0.2 + 0.07S)e^{-sl}}{S^3 + 10.385S^2 + 3.85S + 0.81949Se^{-sl}} \qquad (15)$$

Equation (15) was used to find the Bode plots of Figures 13 and 14. A 5th order Pade approximation was used for the various delays. Figure 13 shows the Bode plot of the engine generator system without the delay. In this case the gain margin is infinite. The Bode plot for the system with a delay of 25.0 ms is shown in Figure 14. The gain margin is now 5.346. A plot of system gain margin vs. system delay is shown in Figure 15.

Fuzzy logic is much like classical PD control in that it uses proportional and derivative terms to determine the control signal. The difference lies in the nonlinear nature of the fuzzy controller. In fuzzy controllers the effective gain is not fixed as it is in classical control schemes, but instead it varies with the system dynamics. When a disturbance occurs, fuzzy controllers can respond with gains outside their stable range for short periods of time and still maintain overall stability as long as the gain changes to its stable range as the system approaches its desired output. The Bode plots of Figures 13-

14 establish limits on the effective gain that the fuzzy controller must satisfy when the system is near its desired output.

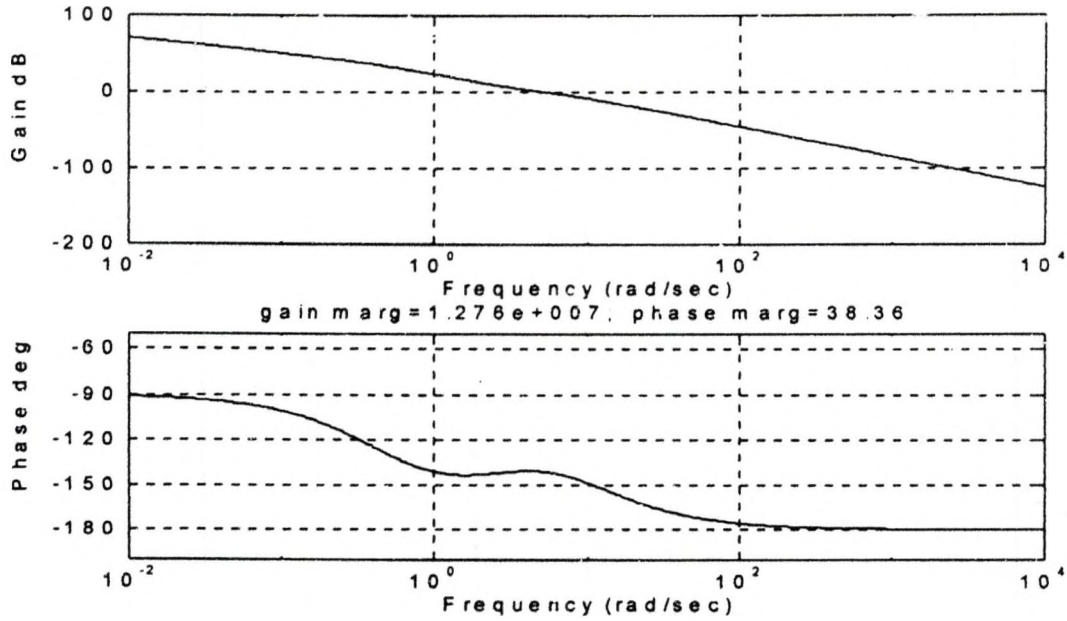

Figure 13. Bode Plot with Zero Delay

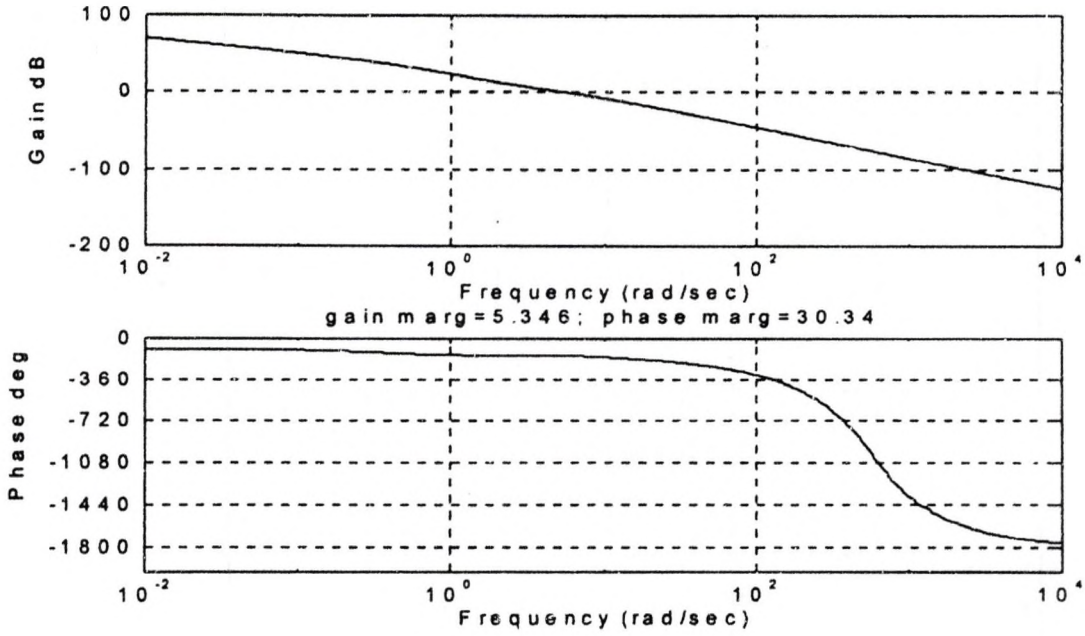

Figure 14 Bode Plot with 25.0 ms Delay

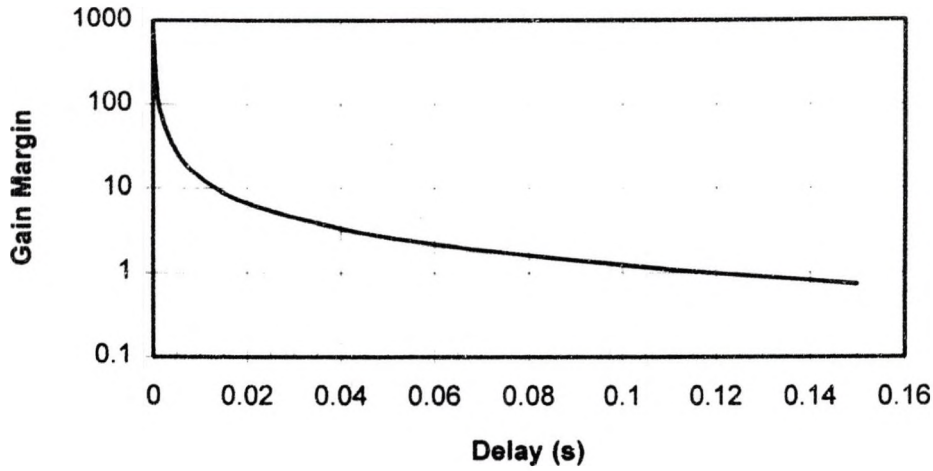

Figure 15. Gain Margin Vs. Delay

For the purpose of reducing the overshoot and oscillations resulting from system delay and the integration effect of the throttle actuator, a supplementary fuzzy controller will be designed to run in parallel with the basic fuzzy controller previously developed. This supplementary controller will only be active when conditions of overshoot are likely to occur. Only two rules are needed to accomplish this goal. The rules are as follows, 1) if the error is slightly positive (i.e. e is P) and decreasing rapidly (i.e. e' is N) then decrease the throttle, and 2) if the error is slightly negative (i.e. e is N) and increasing rapidly (i.e. e' is P) then increase the throttle. Two input membership functions are needed (Negative and Positive). The e and e' input membership functions are shown in Figure 16. Input gains must be used to scale the inputs in order to avoid saturating the controller. Saturation occurs when the input to a membership function is greater than maximum width of the membership function in which case the function saturates to its

maximum value. After saturation occurs the controller will fail to respond properly to system disturbances.

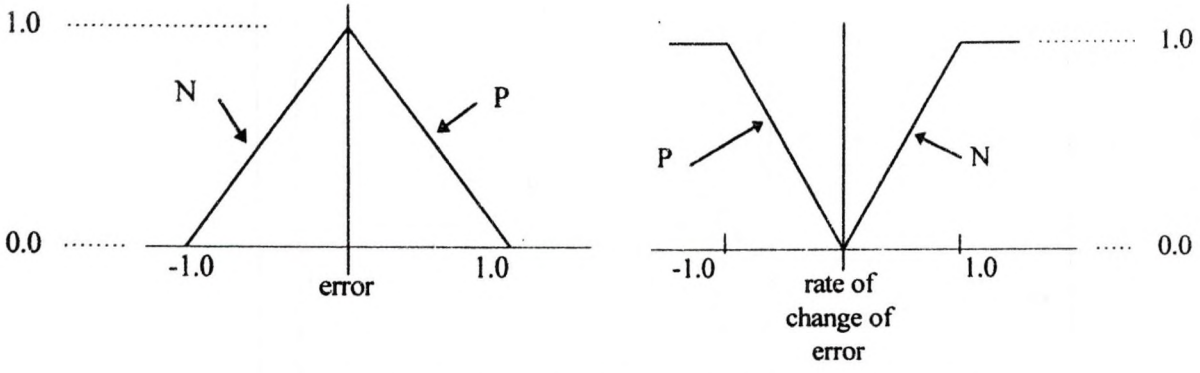

Figure 16. New Controller Input Membership Functions

Different operating conditions as determined by sudden load and speed changes require the shape of the input membership functions to be dynamic in order to maintain system performance. This is handled by adjusting the input gain for the error membership function in response to the largest magnitude of system error resulting from a disturbance. The output gain of this controller must also be adjusted according to the same maximum overshoot criteria described above. Computer simulations were run to determine suitable parameters for the input and output gains in response to various magnitudes of maximum error. Different load magnitudes were applied and the input and output gains of the new fuzzy controller were tuned for the optimal system response. Table 4 summarizes the results.

Load (ft-lbs)	Max Error (rpm)	Input Gain	Output Gain
1	50	0.025	150
3	200	0.005	50

Table 4. Optimal Gains for New Fuzzy Controller

From Table 4, equations 16 and 17 were obtained to determine the proper input gains in response to the maximum error for any disturbance.

$$Input\ Gain = \frac{0.025}{150}(200 - \max\_err) + 0.005 \tag{16}$$

$$Output\ Gain = \frac{150}{150}(200 - \max\_err) + 50 \tag{17}$$

where max_*err* is the maximum speed error resulting from any disturbance. A disturbance will cause the system speed to deviate from its static speed resulting in error. This error is monitored until the controller reacts and begins to bring the system back towards its nominal speed. The point where the rate of change of the error signal after a disturbance is zero is the point where the maximum error occurs. max_*err* is then set to this value and the input and output gains are set from eqn's (16-17). These gains remain fixed until the system error passes through zero at which time the process repeats. This method of setting gains improves the controllers ability to damped oscillations.

The rule table for this fuzzy controller is shown in Table 5. Under most operating conditions the output of this controller will be zero therefore the performance of the system will be determined by the main controller only. Only when overshoot is predicted will this controller influence the system behavior. After the overshoot is suppressed the rate of change of the error signal will be small and the output of this controller will again be negligible.

e \ e'	N	P
N	Z	P
P	N	Z

Table 5. Inference Rule Table

Figure 17 shows the relationship of the secondary controller inference rules to the inference rules of the primary controller. Since the input gains are dynamic the secondary inference rules tend to drift around in a bounded region of the primary controllers rule table.

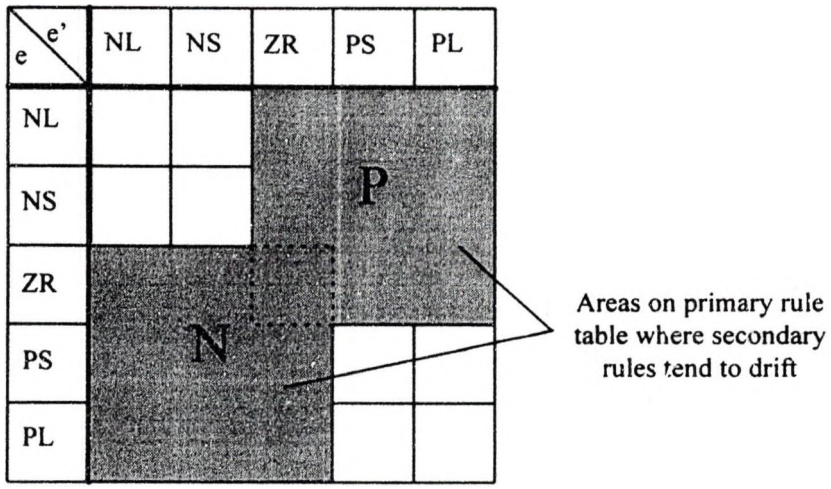

Figure 17. Relationship of Secondary Inference Rules to Primary

3.4 Computer Simulation

A Windows based computer program was written to solve differential equations that describe the systems dynamics. Any number of systems can be simulated and

graphed simultaneously. This program can also be configured to do real time control of a broad range of systems.

The system must be broken down into simple mathematical operations such as addition, multiplication , differentiation and integration. Other dynamic effects such as delays, step inputs and fuzzy controllers are also supported. The parameters of any block can easily be changed and blocks can be connected in any combination. Total simulation time and simulation step size are also easily changed.

Dynamic simulation takes place by repeatedly updating and latching each block in the model at time intervals equal to the simulation step size. The update function causes each block in the simulation to first determine its input from the blocks connected to it and then calculate its next output value. This output value is temporarily stored internally until each block is updated. After every block is updated the latching process begins. In this process the temporary value that is stored internally from the last update is moved to the blocks output, thereby preparing the system for another update. This procedure is repeated until the total simulation time expires or the user terminates the process.

Blocks can be connected to other simulation blocks or they can be connected to an external source. External sources could be any means of exchanging information externally such as A/D and D/A boards or direct digital inputs or outputs. This feature makes the software very powerful in its ability to not only develop control schemes through computer simulation, but also to do real time control and testing.

The accuracy of this program was tested by simulating the engine model developed in Chapter 2. A simulation was run with a step input of load torque of 1.5 ft.lbs at 5 seconds and a step input of throttle of 6 degrees at 15 seconds. The results are compared with the identical simulation run on TUTSIM, see Figure 18. The results prove the accuracy of the new program.

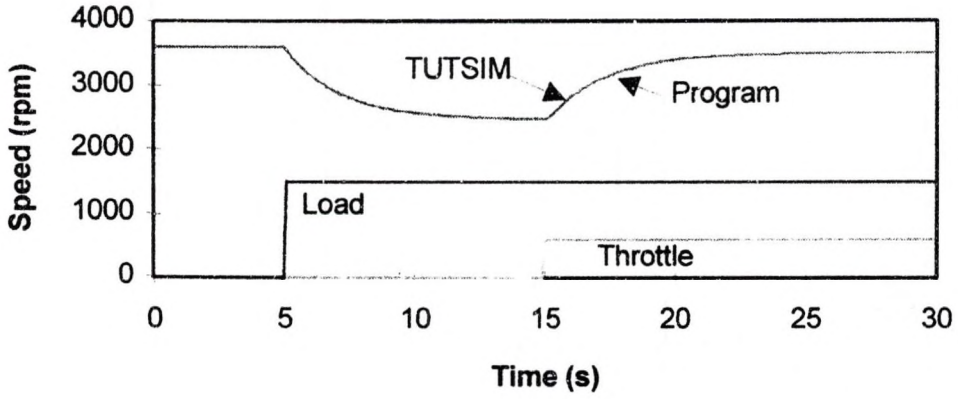

Figure 18. Validation of Computer Program

Figure 19 shows the main window of the dynamic simulation program. The main window contains six sub-windows each displaying a different aspect of the simulation. The Error Memberships window shows the input membership functions, the input gain and an arrow indicating the current input. Similar windows show the error and derivative membership functions for the basic and advanced fuzzy controllers. The Fuzzy Output window shows the output memberships, the output gain and the instantaneous crisp output signal (CS). The input and output gains can be changed while the simulation is running by simply clicking the mouse on the corresponding arrows. The Time Response window shows the time response of the system(s) being simulated. This program can

simulate several systems concurrently allowing real time evaluation of different control schemes.

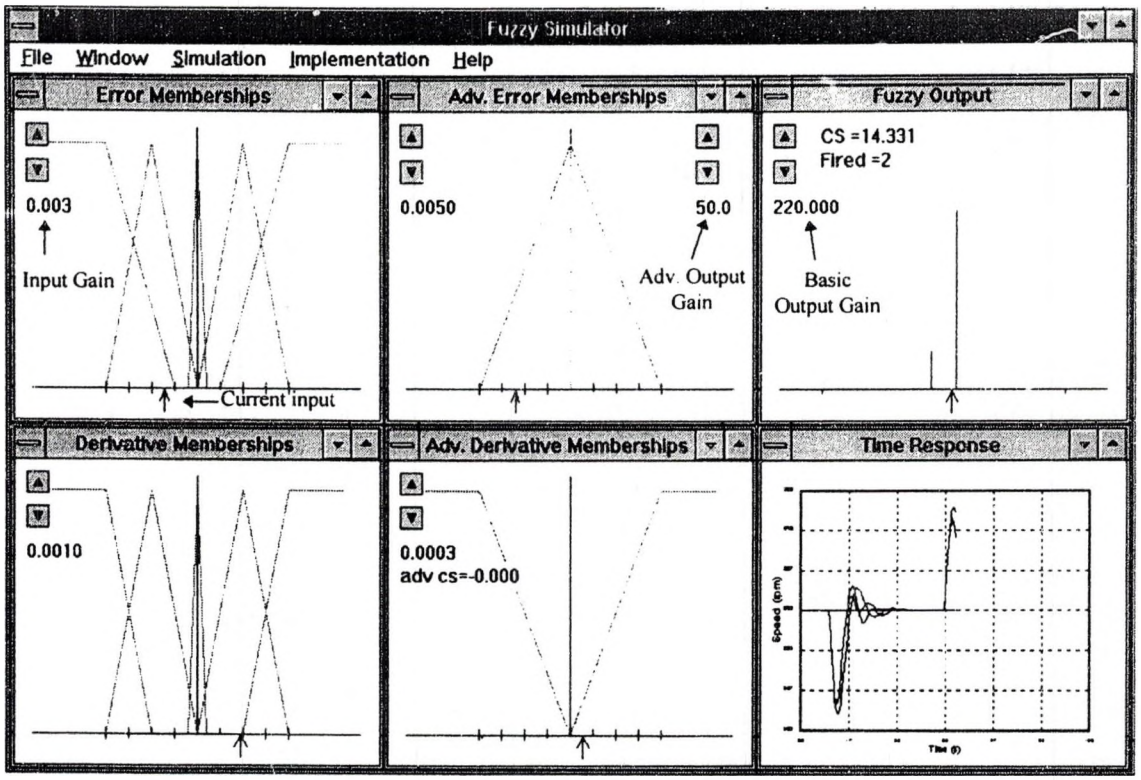

Figure 19 Main Window of Dynamic Simulation Program

Several simulations were run to evaluate the advanced fuzzy controller's performance in comparison with a PID and a standard fuzzy controller. Each simulation was started with the system running at nominal speed (3600 rpm) and no load. A step increase in load was then applied and each system was allowed to return to its nominal speed at which time the load was removed and the system was again allowed to stabilize. A set of simulations were conducted with different values of engine time delays to evaluate each controllers robustness to delays. Another set was run with a constant delay of 25 ms and the engines parameters perturbed to evaluate robustness to parameter changes.

Each simulation run was made with the three control schemes active for comparison. Figures 20 (advanced fuzzy), 21 (basic fuzzy) and 22 (PID) show the block diagrams used to simulate each of the different control schemes.

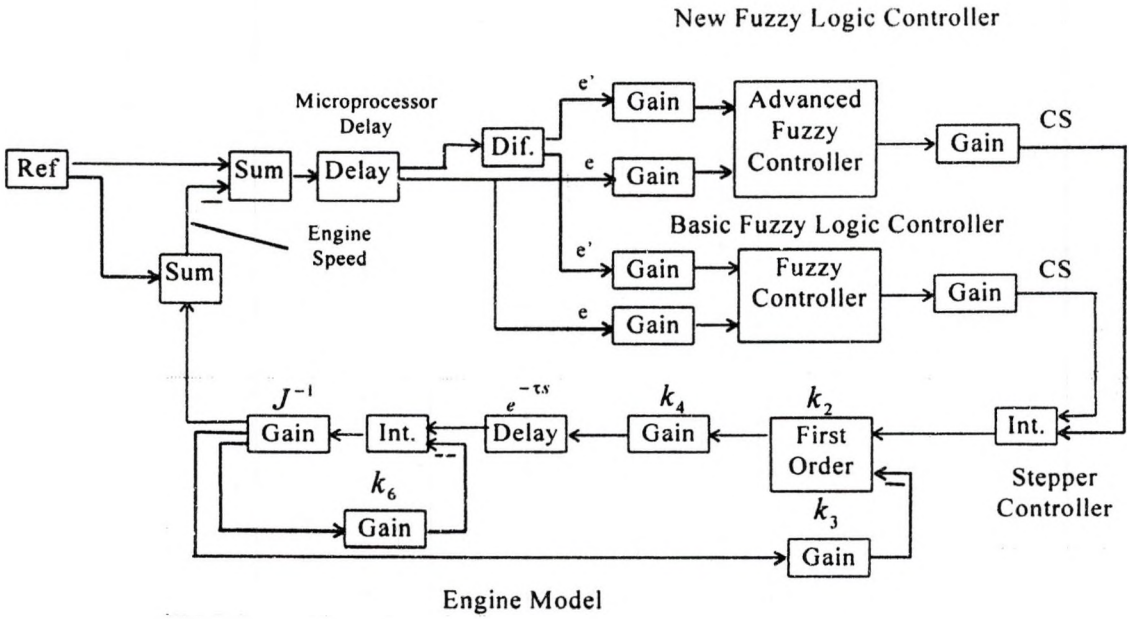

Figure 20. Advanced Fuzzy Logic Control

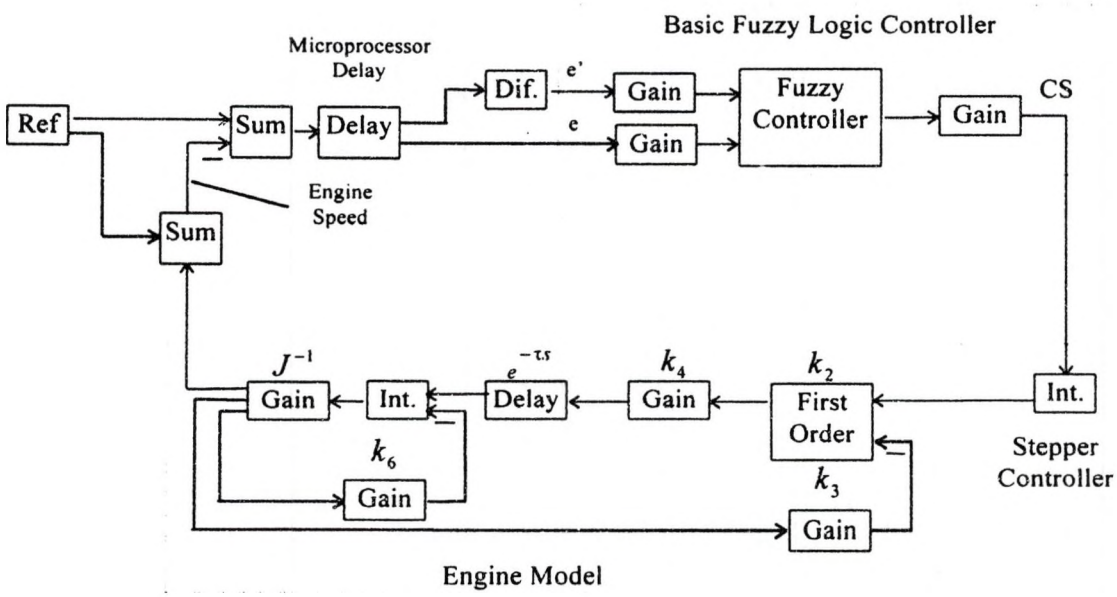

Figure 21. Basic Fuzzy Logic Control

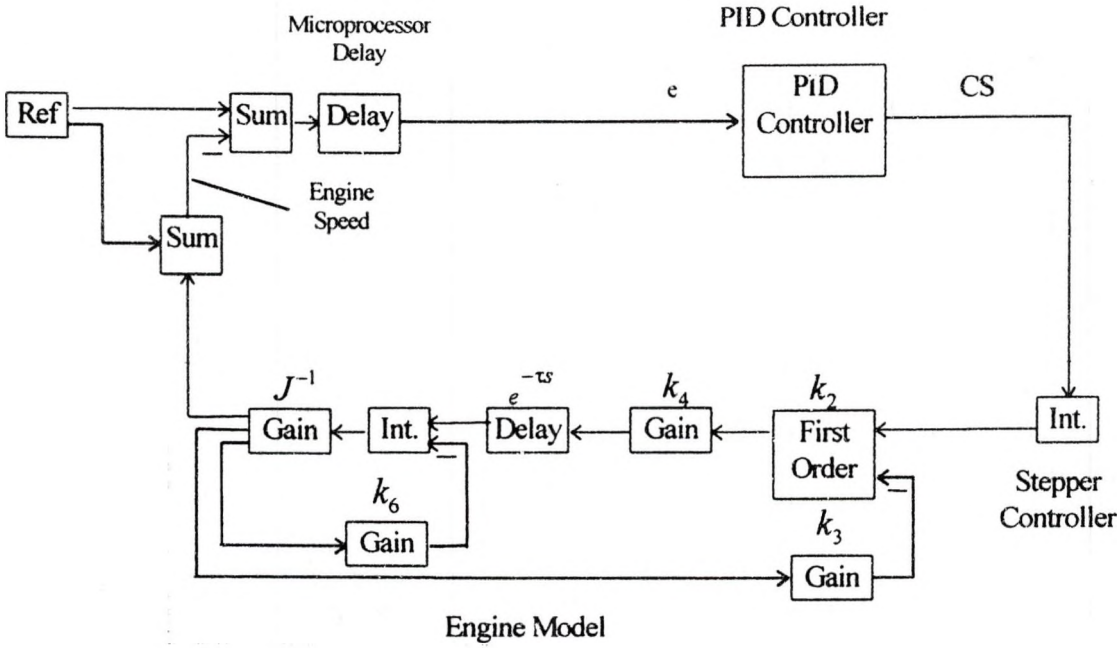

Figure 22. PID Control

Table 6 shows the various gains used by each controller. Note that Ki for the PID controller is set to 0.0 to avoid a double integrator effect of the throttle actuator implementation and the PID controller.

Gain	Adv. Fuzzy	Basic Fuzzy		Gain	PID
e	0.005 to 0.025	0.003		Kp	0.2
e'	0.0003	0.001		Ki	0
output	50 to 150	220		Kd	0.07

Table 6. Controller Gains

The first set of runs determine which control scheme was most robust to a change in system delay. The delay inherent to the Honda engine was determined to be between 16.7 and 33.3 ms the average being 25 ms. Runs were made with the delay at the

minimum (16.7 ms), the average (25 ms), and the maximum (33.3 ms). Each run was started with the engine running at nominal speed with no load. A load of 3.0 ft-lbs is then applied at t=1.0 s and removed at t=5.0 s. The system response is recorded for each of the three delays. The results are shown in Figures 23, 24 and 25. It can be seen that the new fuzzy controller reduces overshoot in comparison with PID or standard fuzzy control while the three controllers stabilize in about the same time. The fuzzy control schemes seem to exhibit more swings than the PID. This can be partially attributed to the small number of input membership functions for these controllers. Adding more input membership functions allows for more precise tuning and increased performance. It should be noted that the abbreviation P.V. Fact. in the following figures represents the parameter variation factor. This is the scaling factor used to vary the parameters of the engine model to show robustness of the controllers to parameter variations.

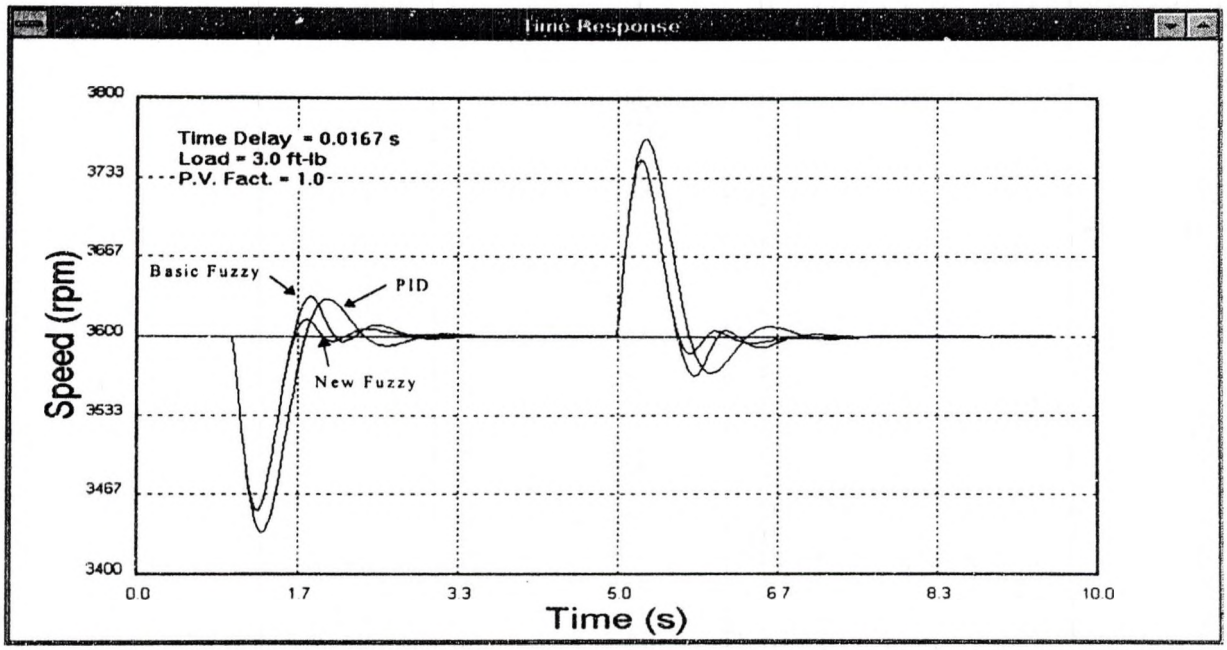

Figure 23 System Response with Delay of 16.7 ms

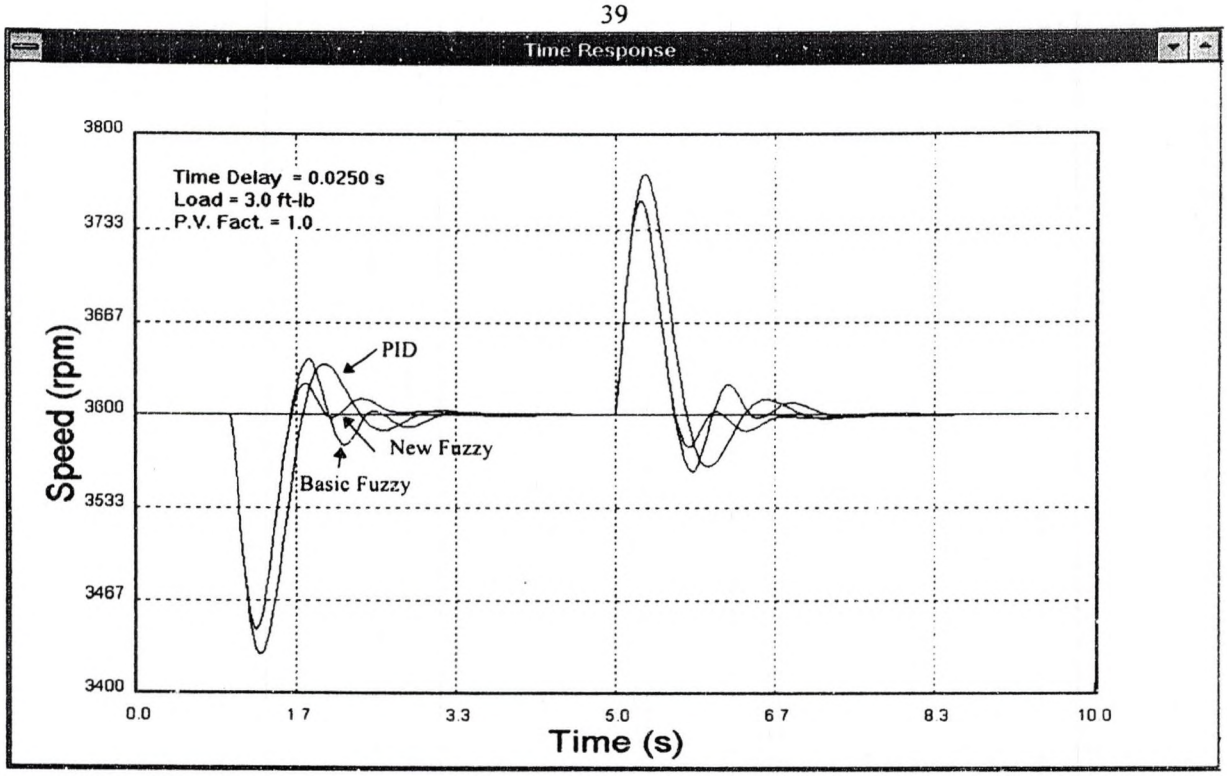

Figure 24 System Response with Delay of 25.0 ms

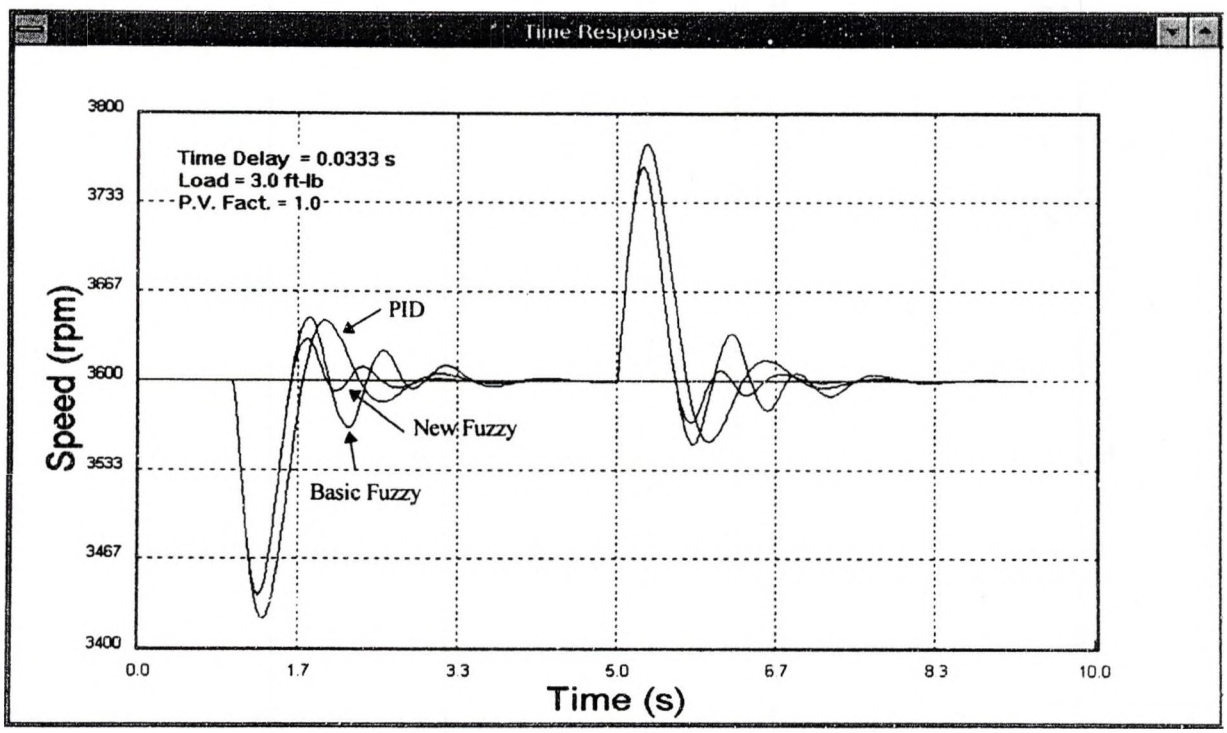

Figure 25 System Response with Delay of 33.3 ms

The next set of runs are to show robustness of the three controllers in response to parameter changes in the engine model. Runs are made with engine parameters scaled by ± 10%. Figure 26 shows the system response with all engine parameters scaled by +10% and Figure 27 shows the response with parameters scaled by -10%.

It can be seen that the new controller is superior to other controllers in both cases. This shows increased robustness to changes in parameters.

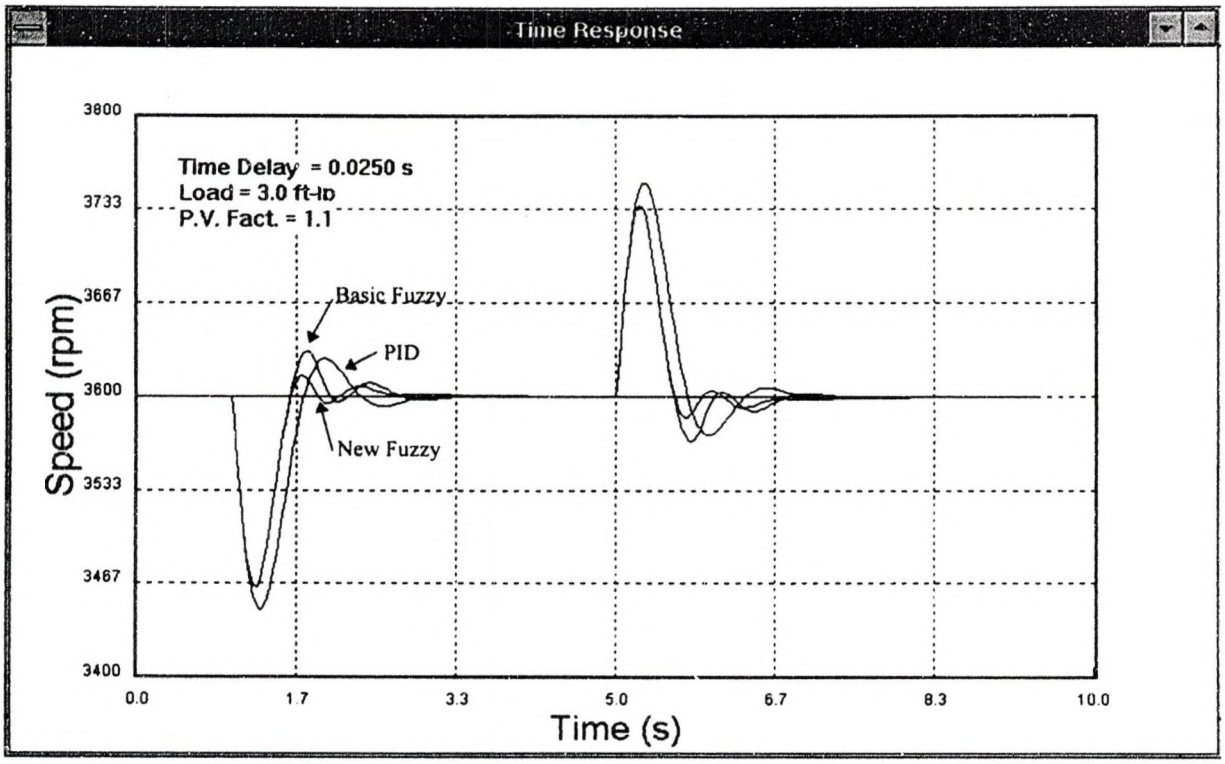

Figure 26 System Response with +10% Engine Parameters

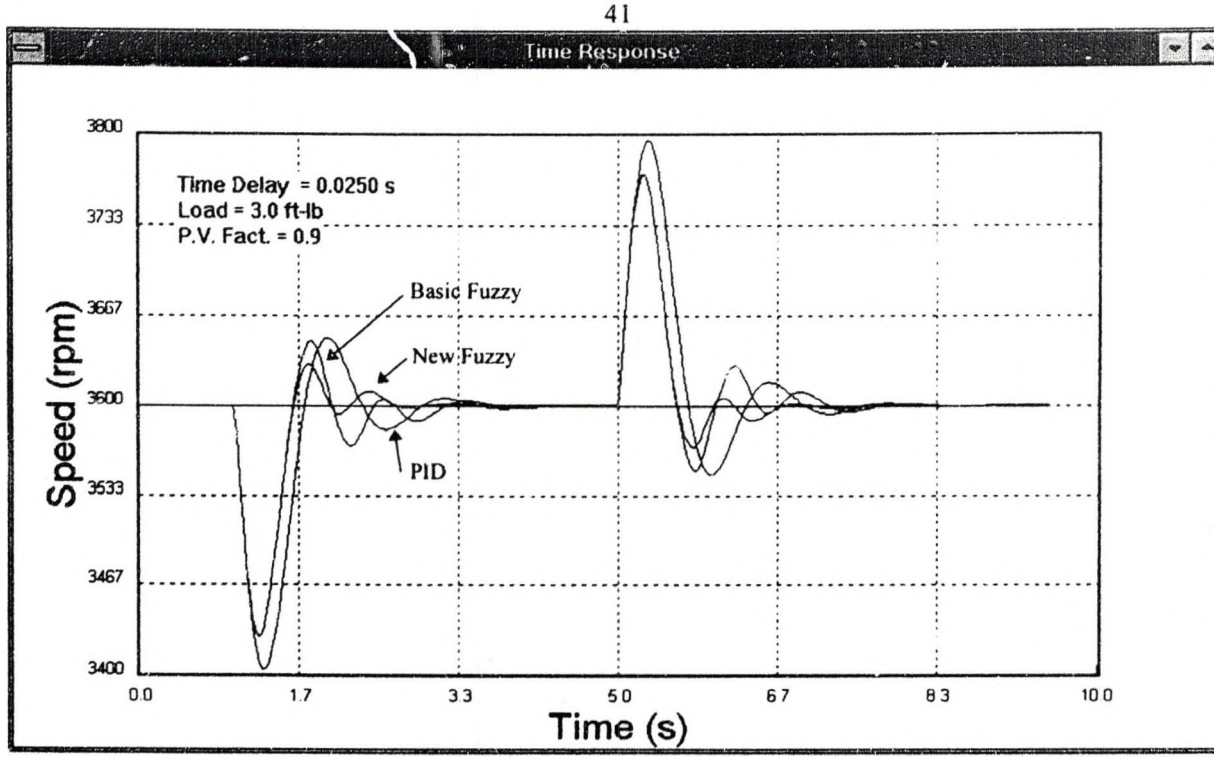

Figure 27 System Response with -10% Engine Parameters

Figure 28 shows the control signals for each of the three controllers. The control signals along with the error signal for the new fuzzy controller are graphed in response to a step input of load. The simulation is identical to the one in Figure 24 (i.e. 3.0 ft-lb load and 25.0 ms time delay). It can be seen that the PD controller produces the smoothest control signal. Both fuzzy controllers have identical control signals until the point where the new fuzzy controller predicts overshoot and responds by increasing the control signal. The fuzzy control signals are more irregular than the PD control signal. This is partially due to the small number of input membership functions. Increasing the number of membership functions tends to smooth out the control signal and improve the dynamic response of the system.

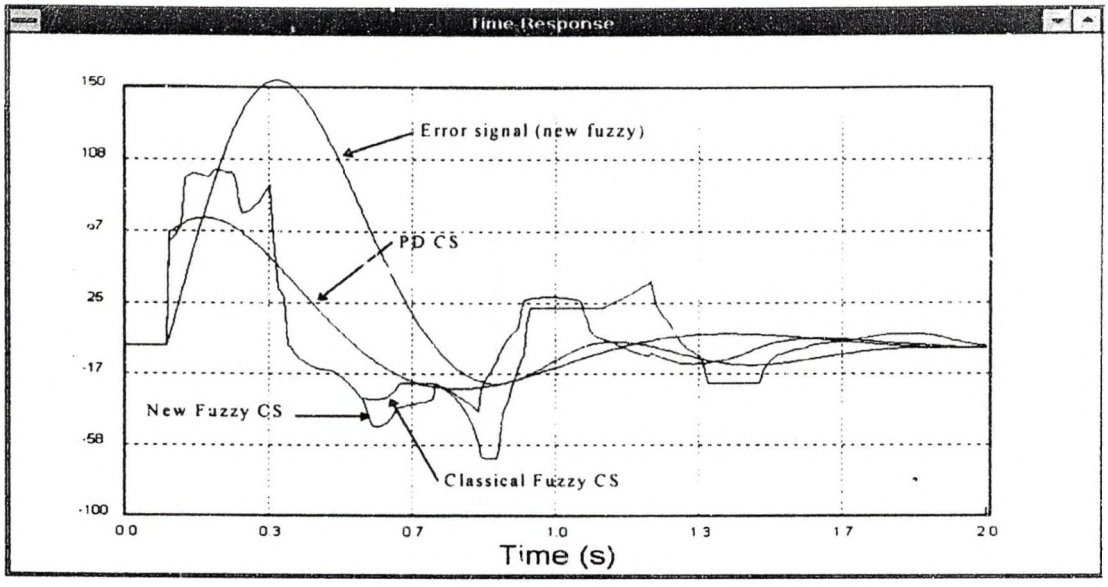

Figure 28 Control Signal Analysis

The following files are associated with the simulation part of this program.

ADC.CPP and DIG.CPP

These files contain the code to call the analog to digital (ADC.CPP) and digital output (DIG.CPP) cards. They are used only when the program is used to control an external system. ADC.H and DIG.H contain the class declarations for these files.

BLOCKS.H

This files contains the definitions of each of the available simulation blocks. The possible blocks are pulse, sum, gain, delay, integrator, differentiator and fuzzy controller. Several other predefined groups of blocks are also available including first order and PID

MODEL.H

This files contains the actual declaration and connection of the blocks to be simulated.

SIMULATE.CPP

This file contains the code that sets up the simulator and runs the simulation. Functions that connect blocks and run updates and latches are included here.

SIMULATE.H

Contains the class declarations for the file SIMULATE.CPP.

STEP.CPP

This is the main file that contains the Windows interface. All the window procedures and graphics are done in this file. This file also contains the code that calls the simulator and fuzzy controller code.

TIMER.CPP

This file contains the timer class to keep track of the current simulation time and create time delays.

TIMER.H

Contains the class declarations for TIMER.CPP.

4. IMPLEMENTATION

In this chapter the fuzzy controller designed in the previous chapters will be implemented to control the speed of the Honda generator. The C++ program described in chapter 3 will be used to realize the various control schemes and record data in the lab. A/D input and digital output channels will be used to interface the control program to the Honda system. After describing the various components, several tests will be run to determine the performance of the new controller compared to standard control schemes.

4.1 Hardware

The entire system can be broken down into four basic subsystems; engine-generator, computer system, throttle actuator and speed sensor. The engine generator is the Honda EM3500S portable generator. The components are connected as shown in Figure 29. Figure 30 shows a picture of the entire lab setup.

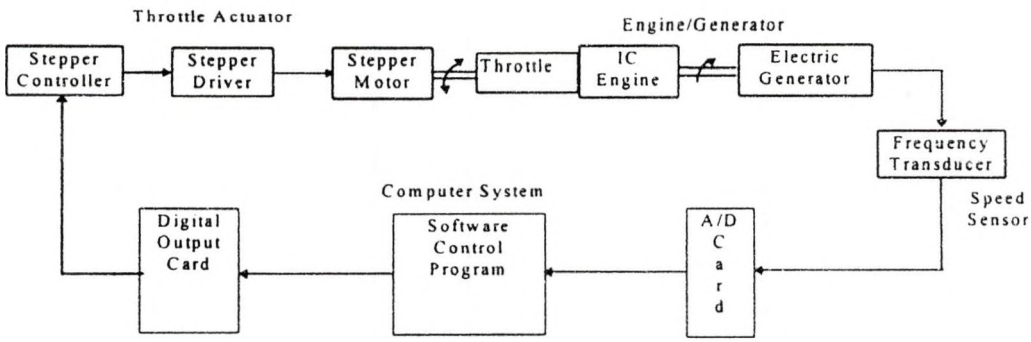

Figure 29. System Components

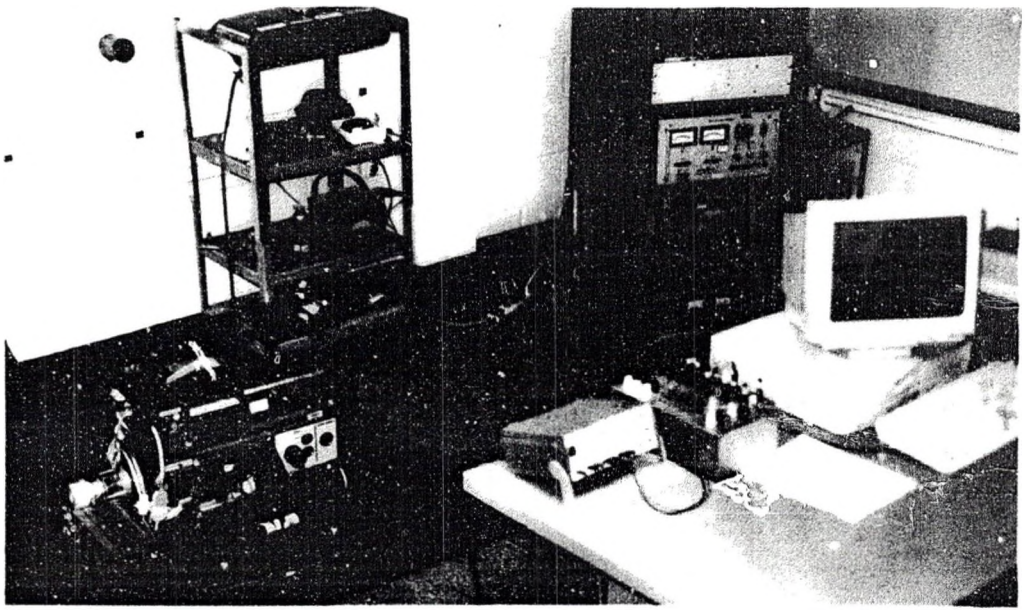

Figure 30 Lab Implementation

Engine/Generator

This system consists of a Honda EM3500S portable generator with its original mechanical speed control system disabled. The generator is rated at a peak output of 3.5 kVA and nominal output of 3.0 kVA. The engine is a 5.0 hp 4 cycle Honda gasoline engine. The generator is a single phase synchronous machine. The rated speed is 3600 rpm which delivers electricity at 120/240 volts and 60 Hz. Voltage control will be handled by the original Honda voltage regulator.

Computer System

The computer system consists of an IBM compatible 486-33 MHz personal computer with several I/O cards. The A/D card is a Cyber Research DAS-1601 with 8

± 10 volt A/D channels. This card will read the analog output of the speed sensor and convert it into a digital number for use by the software. The DAS-1601 is also equipped with a 16 bit counter/timer chip that will be used for system timing. The digital output card is a Cyber CYDIO-96 card with 96 digital I/O channels. Eight of these channels were configured as digital outputs and will be used to send the speed control signal to the stepper controller. The software control program was written and compiled with Borland C++ 3.1 for windows. The entire control loop including the A/D conversion and digital output takes approximately 2 ms.

Throttle Actuator

A Mycom high resolution stepper motor system was chosen as the throttle actuator. The system consists of a model UPS 566MB five phase stepper motor and driver and a model MAC-300 programmable microstepping motor controller. The system is capable of 0.18 degree steps (2000 steps/rev). The controller is programmed using a BASIC like language and is very flexible to different implementations. The controller is programmed to accept an 8 bit digital input to control the speed and direction of the stepper motor. This is done by writing a simple program to interface the controller to the PC. The code and explanation is as follows:

```
@0                          ;assign controller #
$WR.PROG0                   ;assign program #
ACC.RATE.A=200000           ;set acceleration rate
DEC.RATE.A=200000           ;set deceleration rate
L0                          ;beginning of control loop marker
LREG0=0                     ;reset long register 0 (new speed)
SREG0=IN.A                  ;put 8 bit digital input A into short register 0
IF SREG0<128 THEN L1        ;check MSB (direction), branch to L1 if not set
GOTO L2                     ;else branch to L2
```

```
L1                      ;marker for clockwise move section
LREG0=SREG0             ;move speed to lreg0
IF SREG5==1 THEN L5     ;branch if direction did not change
NOR.STOP.A              ;must issue stop before direction change
L5
RUN.VEL.A=LREG0         ;set axis A speed
VEL.MOV.A               ;move axis A at preset speed
SREG5=1                 ;set previous direction register
GOTO L0                 ;repeat control loop
L2                      ;counterclockwise section (same as clockwise)
SREG0 = SREG0/2         ;shift 1 bit to right
LREG0=SREG0
IF SREG5==0 THEN L6
NOR.STOP.A
L6
RUN.VEL.A=LREG0
VEL.MOV.-A
SREG5=0
GOTO L0
END
```

The driver for the stepper motor is a model UPS 500 and it accepts step and direction inputs from the MAC-300 controller. This driver converts the logic signal from the MAC-300 stepper controller to the 5 phase signal needed to rotate the phases of the stepper motor.

Speed Sensor

An Ohio Semitronics, Inc. model FTA-006-004X5 frequency transducer was chosen to measure the speed of the engine. This device is capable of accurately ($\pm 0.5\%$) measuring frequencies from 0-425 Hz of sinusoidal voltages from 80-150 volts. The output is a 0-5 volt DC signal proportional to the frequency. This signal can be read directly into the computer via the A/D card.

4.2 Realization of Controller

The program outlined in section 3.4 was adapted to do real time control of the Honda generator. The original program was written to run in the Windows environment but yielded unsatisfactory results with real time control. The problem resulted from Windows periodically seizing control of the system to perform system oriented functions causing the control program to be inactive for relatively long periods of time. This inactivity decreases performance to unacceptable levels. The solution was to implement the program in the DOS environment thereby guaranteeing consistent sample times. The output of the Honda generator was read via an A/D block and the control signal was sent out via a digital output block. The rest of the program remained the same except that the system time was calculated from a timer before each iteration. The same input and output gains and inference rules were used in the lab implementation as were used in the computer simulation. A file block is used to save simulation data so it can be analyzed and displayed after the simulation is finished. The basic block diagram for the system is given in Figure 31.

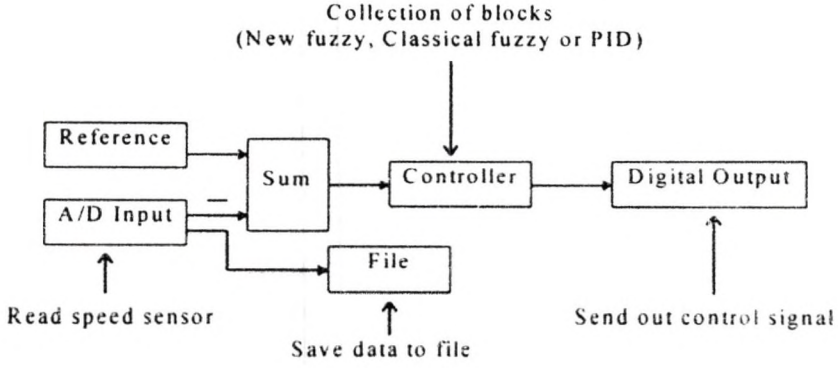

Figure 31. Basic Block Diagram for Implementation

4.3 Experimental Results

Several runs were made to analyze the performance of the advanced fuzzy controller in comparison to the standard fuzzy and the PID controller. The first runs attempt to show the performance of each of the control schemes in response to a disturbance. Figures 32-34 show the classical fuzzy, new fuzzy and PD controllers response to a step increase and decrease in load. Each system was run at 3600 rpm and an electrical load of 1500 watts (2.93 ft.lbs torque) was applied. The systems were allowed to return to 3600 rpm at which time the load was removed and the systems were again allowed to stabilize. A resistive load rack was used as the electrical load.

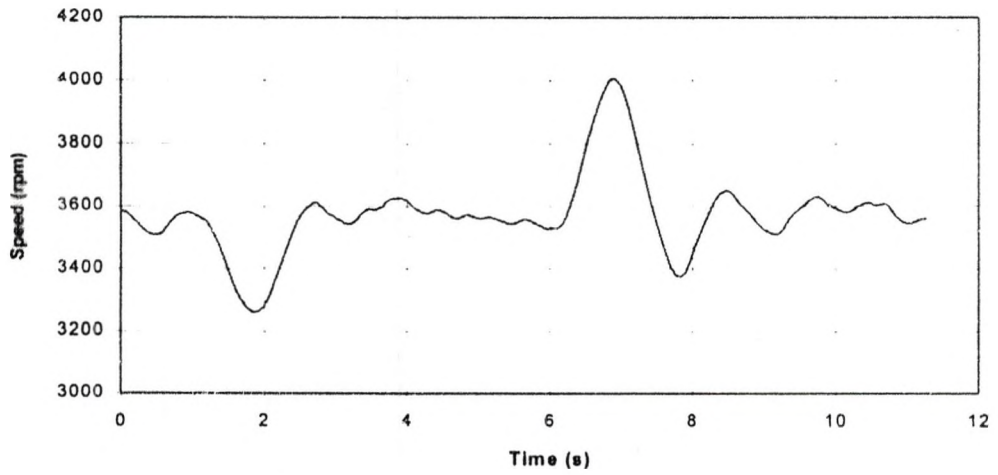

Figure 32 Classical Fuzzy Control

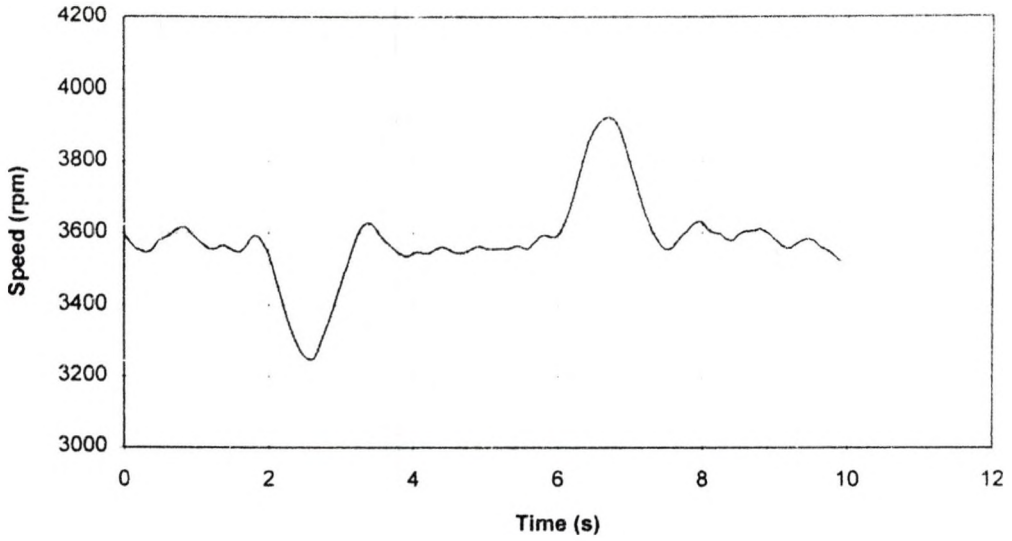

Figure 33 New Fuzzy Control

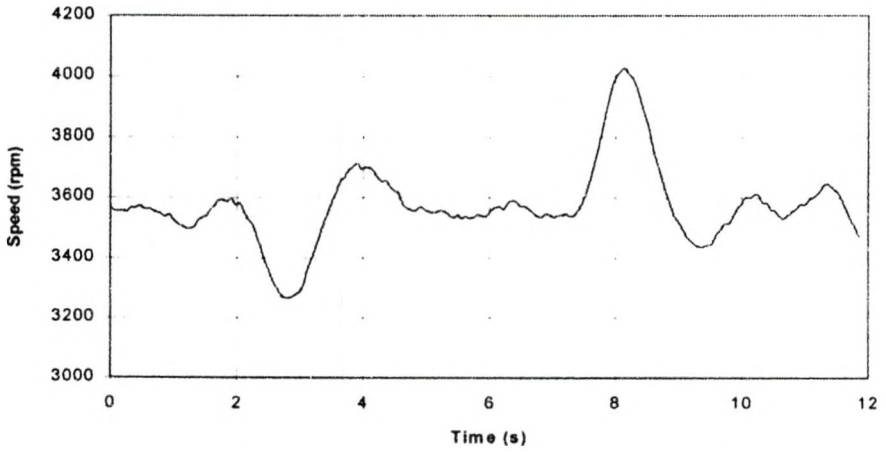

Figure 34 PD Control

It can be seen that the new fuzzy controller exhibits slightly less oscillations than either the traditional fuzzy or PD controllers. The results are difficult to verify precisely due to the large amount of oscillations during steady state operation. These oscillations are attributed mostly the time constants of the frequency transducer and the stepper motor

and its controller. Electrical noise and sensor error also play a role in some of the oscillations.

Another set of runs were made to test each of the controllers robustness to parameter changes. To simulate parameter changes the engine was run at 3000 rpm (50 Hz). It is unknown to what degree each parameter changes but the nonlinear nature of processes such as combustion and intake manifold dictates some variation at different operating speeds. Figures 35-37 show the traditional fuzzy, new fuzzy and PD controllers response to a sudden addition and removal of 1500 watts of electrical load.

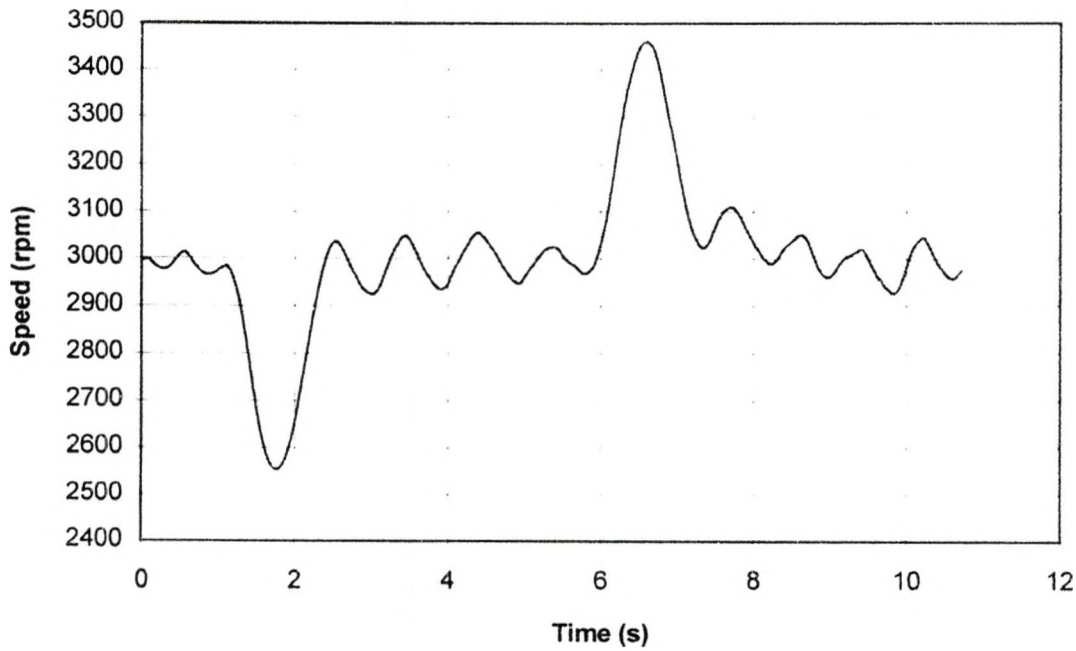

Figure 35 Classical Fuzzy Control

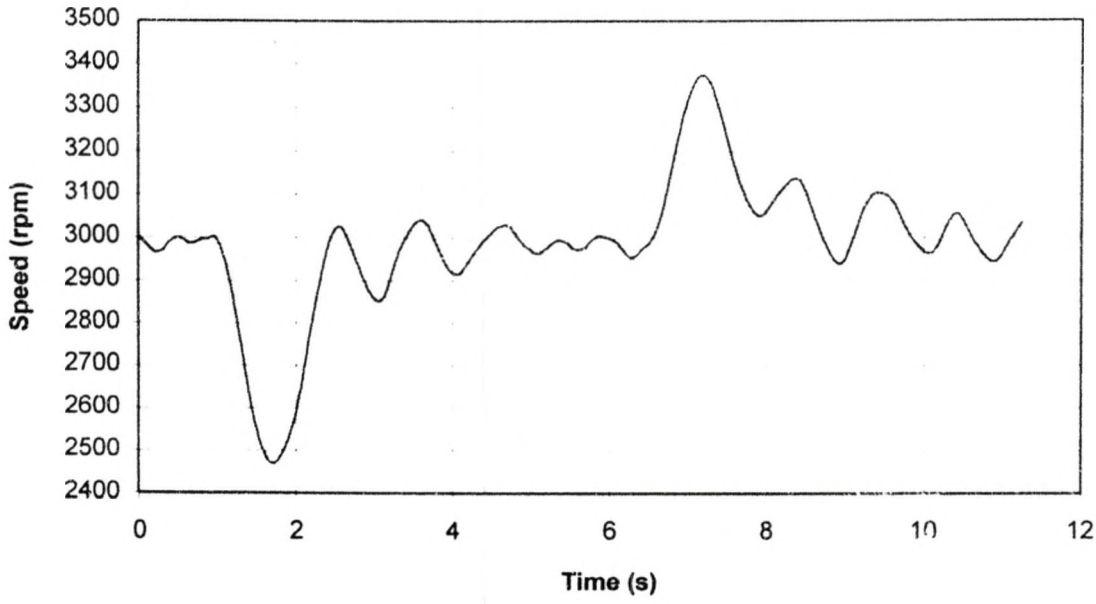

Figure 36 New Fuzzy Control

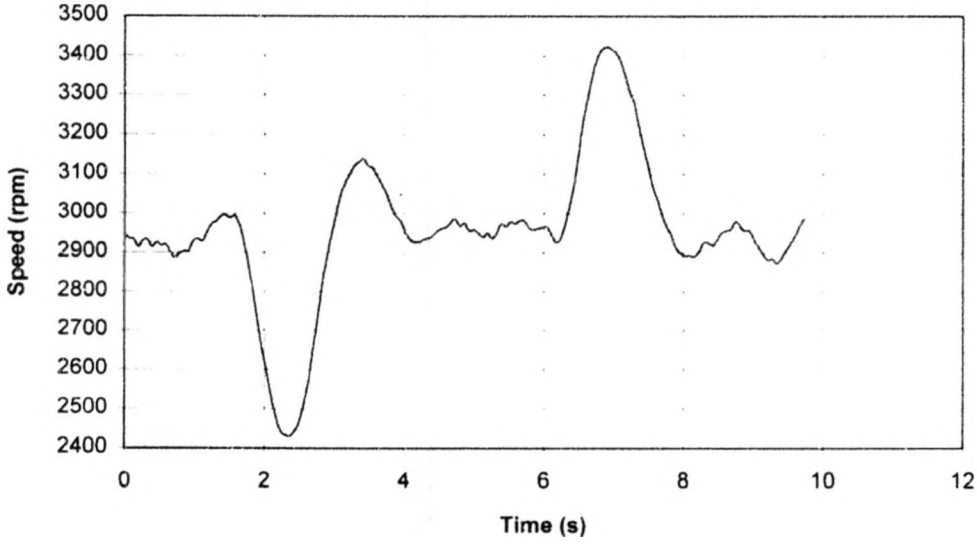

Figure 37 PD Control

The results of this run are inconclusive as to which response is best. The PD controller dampens out the oscillations the quickest but it suffers from the most overshoot. The traditional fuzzy and new fuzzy controllers show close responses. Again the large amount of oscillations at steady state accounts for the inability to distinguish performances. These runs do show that the fuzzy controllers are more robust to parameter variations because they exhibit less overshoot than the PD controller.

Figure 38 shows the dynamic performance of the Honda generator with it's original mechanical speed controller in response to an application of 1500 Watts (2.93 ft.-lbs) of load. It can be seen that the mechanical speed controller is very fast in response to changes in load with very little oscillation. A steady state error of slightly greater than 100 rpm exists when the load is applied. This error is undesirable for equipment that is not tolerant of frequency deviation.

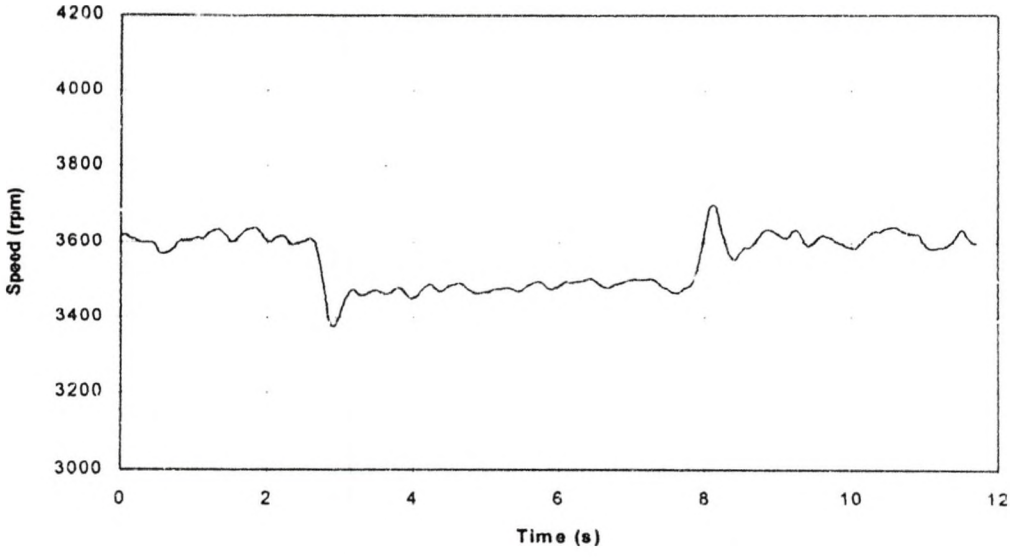

Figure 38 Mechanical Speed Control

5. CONCLUSION

Internal combustion engines exhibit characteristics such as nonlinearities and time delays that make them challenging to control. Classical control methods such as Proportional Integral Derivative (PID) are not ideally suited to these challenges due to their constant (linear) gains. Modern methods such as fuzzy logic are more suited to engine control because of their nonlinear nature. In this thesis a fuzzy logic controller for an engine-generator system was designed and implemented in the laboratory.

The first step in the development of a control scheme is to create a mathematical model of the system to be controlled. Several tests were conducted to determine various engine parameters such as manifold dynamics, combustion and rotational dynamics. These parameters were then fine tuned via computer simulation to establish an accurate model.

The next step was to design the controller. A standard five membership fuzzy controller with inputs of error and error derivative was chosen as a starting point. The initial inference rules were chosen based on how the system should respond to various engine operating conditions. A computer program was written to realize and test the new controller with the dynamic engine model. During this phase the controller's inference rules were fine tuned to achieve the best performance.

It was found that the classical fuzzy controller exhibited overshoot and oscillations. To remedy this problem a dual fuzzy controller was designed to run in parallel with the previously designed fuzzy controller. This new controller was tuned to anticipate when conditions of overshoot are likely to appear and respond by introducing an appropriate control signal to reduce both overshoot and oscillations.

The new controller was compared to both classical fuzzy control and PID control. Each controller's robustness to delays was tested by running simulations with various engine time delays. Next each controller's robustness to parameter variations was determined by running simulations with various engine parameters. It was found that the performance of the new fuzzy controller is superior to both the classical fuzzy and the PID.

After the control scheme was tested and verified via computer simulation it was implemented on the physical system in the lab. Several modifications to the existing engine-generator were made including the removal of the existing speed controller, the addition of a throttle actuator and the addition of a speed sensor. The previously mentioned computer program was modified to do real time control of the engine-generator.

Several tests were then conducted in the lab to determine the performance of each controller on the actual system. In the first test each controller's performance was recorded in response to an application of load to the generator with the engine running at

its rated speed of 3600 rpm (60 Hz). The next test was to simulate a change in parameters, it involved the applying load to the system at 3000 rpm (50 Hz).

The results of these tests demonstrated that the PID is slightly lacking in performance compared to fuzzy control. Comparison of classical and new fuzzy control was largely inconclusive as to which controller achieved higher performance. The main reason for this inconclusiveness was due to a significant degree of speed oscillation under static conditions. These oscillations are probably due to the slow response of the throttle actuator and a noisy speed sensor. With improved hardware results similar to the computer simulations would be expected.

The implemented control schemes developed in this thesis provide valid practical engine control options. The three schemes are stable and they exhibit acceptable dynamic and static performances. The developed controllers are slightly slower than the original mechanical speed controller but are more flexible and do not suffer from steady state error as the mechanical controller does. The fuzzy controllers may be favored over PID for the ease of tuning and improved dynamic performance.

APPENDICES

APPENDIX A

C++ Source Code for the Windows Simulator Program

```
//FILENAME: ADC.CPP
//BY:     Craig Jensen

# include "adc.h"

adc::adc ( )
 {
  int i;
  buffer_sum = 0.0;
  for (i=0;i<ave_num;++i)
     {
     outportb ( ADC_Base_Addr+2,byte( 4 *16)+4);   //start A/D
     outportb ( ADC_Base_Addr+0, 4 );   //start A/D
     while ( inportb ( ADC_Base_Addr + 8 ) >= 128 );   //MSB is EOC bit
     low_byte  = inportb ( ADC_Base_Addr + 0 );
     high_byte = inportb ( ADC_Base_Addr + 1 );
     result =  ( high_byte  * 16.0) +  ( low_byte / 16.0);
     buffer[i]=(((float(result) - 2048.0 ) / 2048.0 ) * Voltage_Range*5100.0);
     buffer_sum += buffer[i];
     }
  buffer_index = 0;
  outportb( ADC_Base_Addr + 0x0B, 0x01 );   //select gain of 10 for A/D input
 }

adc::~adc ( void )
 { }

float
adc::get_data ( byte channel )
 {
  outportb ( ADC_Base_Addr+2,byte( channel *16)+channel);   //start A/D
  outportb ( ADC_Base_Addr+0, channel );   //start A/D
  while ( inportb ( ADC_Base_Addr + 8 ) >= 128 );   //MSB is EOC bit
  low_byte  = inportb ( ADC_Base_Addr + 0 );
  high_byte = inportb ( ADC_Base_Addr + 1 );
  result =  ( high_byte  * 16.0) +  ( low_byte / 16.0);
  buffer_sum -= buffer[buffer_index];
  buffer[buffer_index] = (((float(result) - 2048.0 ) / 2048.0 ) * Voltage_Range/10.0*5100.0);
  buffer_sum += buffer[buffer_index];
  ++buffer_index;
  if (buffer_index >= ave_num) buffer_index = 0;
  return  buffer_sum/float (ave_num);
   }
```

```
//FILENAME:  ADC.H
//BY:        Craig Jensen


# ifndef  ADC_INCLUDED
# define  ADC_INCLUDED

typedef unsigned char byte;
typedef unsigned int  word;

const float Voltage_Range = 10.0;
const word ADC_Base_Addr = 0x0300;
const int ave_num = 5.0;

# include <dos.h>

class adc
 {
  protected :
        byte channel;
        byte high_byte;   //from AD
        byte low_byte;    //from AD
  word result;

  public :

                adc ( void );
                adc ( byte  req_channel );
                ~adc ( void );
                float  get_data ( byte channel );
 };

# endif
```

```
//FILENAME:  BLOCKS.H
//BY:        Brad Trostad and Craig Jensen

#ifndef BLOCKS_INCLUDED
#define BLOCKS_INCLUDED

#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include "simulate.h"
#include "adc.h"
#include "dig.h"



class sb_pulse : public simulation_block
 {
  protected :

   float   start_time;
   float   stop_time;
   float   amplitude;

  public :

   sb_pulse ( simulator *     sim,
          const char *    new_block_name,
          float           pulse_start_time,
          float           pulse_stop_time,
                                                  float          pulse_amplitude ) :
    simulation_block(sim,new_block_name,0,1.0)
    {
     assert ( pulse_start_time >= 0.0 );
     assert ( pulse_stop_time > pulse_start_time );
     start_time   = pulse_start_time;
          stop_time    = pulse_stop_time:
     amplitude    = pulse_amplitude;
    }

   virtual
   float
   output ( float    sim_time )                              .
    {
     if ((sim_time >= start_time) && (sim_time <= stop_time))
      {
       return amplitude;
                                    }
     return 0.0;
    }

   virtual
   boolean
```

```
    is_ready ( void )
     {
      return TRUE;
         }

 };

class sb_sum : public simulation_block
 {
  public :

                    sb_sum ( simulator *     sim,
          const char *    new_block_name ) :
     simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,1.0)
     {
      // Does Nothing Here...
     }

    virtual
    float
    output ( float     sim_time )
     {
      return input_sum(sim_time);
         }

    virtual
    boolean
    is_ready ( void )
                    {
                            if ( total_inputs < 2 )
                                    {
                                     return FALSE;
                                    }
          return TRUE;
         }
 };

class sb_lim : public simulation_block
 {
  protected :
            float upper_limit;
            float lower_limit;
  public :

                    sb_lim ( simulator *     sim,
        const char *    new_block_name,
        float         up_lim,
        float         dn_lim ) :
     simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,1.0)
     {
      upper_limit = up_lim;
      lower_limit = dn_lim;
```

```
    }

    virtual
    float
    output ( float    sim_time )
     {
      if (input_sum(sim_time) > upper_limit) return upper_limit;
      if (input_sum(sim_time) < lower_limit) return lower_limit;
      return input_sum(sim_time);
            }

    virtual
    boolean
    is_ready ( void )
                    {
                            if ( total_inputs < 1 )
                                    {
                                     return FALSE;
                                    }
          return TRUE;
         }
};


class sb_gain : public simulation_block
{
        public :

                sb_gain ( simulator *    sim,
                const char *    new_block_name,
                float           block_gain ) :
        simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,block_gain)
        {
         // Does Nothing Here...
        }

                virtual
                float
                output ( float   sim_time )
        {
         return (input_sum(sim_time) * output_gain);
        }

                float
                get_gain(void)
                        {
                        return output_gain;
                        }

                void
                set_gain(float new_gain)
                        {
```

```
                        output_gain = new_gain;
                        }

                virtual
                boolean
                is_ready ( void )
        {
         if ( total_inputs < 1 )
                {
                        return FALSE;
                }
         return TRUE;
        }
};

class sb_AD_IN : public latched_simulation_block
 {

 protected :
 adc    adinput;

 public :

                sb_AD_IN (  simulator *     sim,
                        const char *    new_block_name,
         unsigned char   channel ) :
        latched_simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,1.0),adinput (channel)
        {

        }

                virtual
                void
                update ( float   sim_time )
              {
      latch_value = adinput.get_data(4)*5100.0;
      //return 3.0;
              }

                virtual
                boolean
                is_ready ( void )
        {
         return TRUE;
        }
};


class sb_DIG_OUT : public latched_simulation_block
{

 protected :
```

```
dig    digoutput;

public :

                sb_DIG_OUT ( simulator *     sim,
                        const char *    new_block_name,
         unsigned char   channel ) :
        latched_simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,1.0),  digoutput ()
        {
 }

~sb_DIG_OUT(void)
    {
    digoutput.send_out(-20.0);
    }

                virtual
                void
                update ( float   sim_time )
            {
    digoutput.send_out(input_sum(sim_time));
    }

                virtual
                boolean
                is_ready ( void )
            {
            return TRUE;
            }
};


class sb_file : public latched_simulation_block
{

protected :

FILE *out_file;
float next_update_time,update_interval;
const char * filename;
float * x_data;
float * y_data;
dword iter_num,i;

public :

                sb_file ( simulator *     sim,
                        const char *    new_block_name,
          const char  *      outfilename,
          const float        update_int ) :
        latched_simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,1.0)
        {
```

```
filename = outfilename;
next_update_time = update_int;
update_interval = update_int;
i = 0;
iter_num = 0;
 x_data = new float [200];
 y_data = new float [200];
 assert (x_data != NULL);
 assert (y_data != NULL);
    }

~sb_file(void)
        {
        out_file = fopen(filename,"wt");
        assert (out_file != NULL);
        for (dword j=0; j < iter_num;j++)
          fprintf(out_file,"%f,%f\n",x_data[j],y_data[j]);

        fclose(out_file);

        }

                virtual
                void
                update ( float   sim_time )
              {
      if ( (next_update_time <= sim_time) && (iter_num < 200) )
        {
        x_data[i]=sim_time;
        y_data[i]=input_sum(sim_time);
        i++;
        next_update_time += update_interval;
        iter_num++;
        }


      }

                virtual
  boolean
                is_ready ( void )
        {
 if ( total_inputs < 1 )
                 {
                         return FALSE;
                 }
                  return TRUE;
        }
 };


class sb_delay : public latched_simulation_block
```

```
{
protected :

  float *      del_data;
  float        last_time;
  float        time_per_sample;
  float        delay_time;
  int          del_index;
  int          num_stored;
  boolean      del_is_full,delta_time;

public :

  sb_delay ( simulator *     sim,
                const char *    new_block_name,
        float          block_delay,
        int            num_to_store,
        float          block_gain ) :
   latched_simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,block_gain)
   {
    output_value = 0.0;
    latch_value = 0.0;
    del_data = NULL;
        delay_time = block_delay;
        num_stored = num_to_store;
        time_per_sample = delay_time / num_stored;
        last_time = 0.0;
        del_index = 0;
        del_is_full = FALSE;
        delta_time = FALSE;
        // allocate memory for delay block and check for invalid pointer
        del_data = new float  [num_stored];
        assert(del_data != NULL);

       }

                virtual
                void
                update ( float   sim_time )
       {
        // check if the delay time is less than the simulation timestep
        // if it is then set the delay = 0
        if ( (delta_time == FALSE) && (sim_time > 0) )
                  {
                   delta_time = TRUE;
                   if ( (sim_time - last_time) >= delay_time )
                  {
                  num_stored = 0;
                  delete del_data;
                  }
                   else if ( (sim_time - last_time) > (delay_time/num_stored) )
                  {
```

```
                num_stored = int( delay_time / (sim_time - last_time));
                time_per_sample = (delay_time / num_stored)-1;
                delete del_data;
                // allocate memory for delay block and check for invalid pointer
                del_data = new float  [num_stored];
                assert(del_data != NULL);
                }
                                }


                        if (num_stored != 0)
                                {
                                if ( (sim_time - last_time) >= time_per_sample )
          {
          del_data[del_index] = input_sum(sim_time);
          last_time = sim_time;
          del_index++;
          if ( del_index > num_stored-1 )
            {
            del_index = 0;
            del_is_full = TRUE;
            }
          }
        latch_value = del_data[del_index];
        if (!del_is_full) latch_value = 0.0;
        }

      else
        {
        latch_value = input_sum(sim_time);
        last_time=sim_time;
        }


    }

   virtual
   boolean
   is_ready ( void )
    {
     if ( total_inputs < 1 )
      {
       return FALSE;
      }
     return TRUE;
    }
};


class sb_integrator : public latched_simulation_block
 {
  protected :
```

```
    float    last_time;

  public :

    sb_integrator ( simulator *    sim,
                const char *   new_block_name,
                float          block_gain,
                        float          initial_value ) :
     latched_simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,block_gain)
     {
      output_value = initial_value;
      latch_value  = initial_value;
      last_time    = 0.0;
     }

    virtual
    void
    update ( float    sim_time )
     {
      latch_value = output_value + ((sim_time - last_time) * input_sum(sim_time));
      last_time   = sim_time;
     }

    virtual
    boolean
    is_ready ( void )
     {
          if ( total_inputs < 1 )
       {
        return FALSE;
       }
      return TRUE;
     }
};

class sb_lim_integrator : public latched_simulation_block
 {
  protected :

    float    last_time;
    float    lower_limit;
    float    upper_limit;

  public :

    sb_lim_integrator ( simulator *    sim,
                const char *   new_block_name,
                float          block_gain,
                            float          initial_value,
                float          min_val,
                float          max_val ) :
```

```
    latched_simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,block_gain)
    {
     output_value = initial_value;
     latch_value  = initial_value;
     last_time    = 0.0;
     upper_limit = max_val;
     lower_limit = min_val;
    }

   virtual
   void
   update ( float    sim_time )
    {
     latch_value = output_value + ((sim_time - last_time) * input_sum(sim_time));
     if (latch_value > upper_limit) latch_value = upper_limit;
     if (latch_value < lower_limit) latch_value = lower_limit;
     last_time   = sim_time;
    }

   virtual
   boolean
   is_ready ( void )
    {
        if ( total_inputs < 1 )
      {
       return FALSE;
      }
     return TRUE;
    }
};


class sb_differentiator : public latched_simulation_block
 {
  protected :

   float  last_time;
   float  last_input;

  public :

   sb_differentiator ( simulator *    sim,
               const char *   new_block_name,
               float          block_gain ) :
       latched_simulation_block(sim,new_block_name,MAX_BLOCK_INPUTS,block_gain)
    {
     output_value = 0.0;
     latch_value  = 0.0;
     last_time    = 0.0;
     last_input   = 0.0;
    }
```

```
    virtual
    void
    update ( float    sim_time )
     {
      float  new_input;

      if ( sim_time <= 0.0 )
       {
        latch_value = 0.0;
       }
      else
       {
            new_input  = input_sum(sim_time);
        latch_value = (new_input - last_input) / (sim_time - last_time);
        last_input  = new_input;
        last_time   = sim_time;
       }
     }

    virtual
    boolean
    is_ready ( void )
     {
      if ( total_inputs < 1 )
       {
        return FALSE;
       }
      return TRUE;
     }
  };

class sb_first_order
  {
   protected :

    sb_integrator  fio_int;
    sb_gain        fio_ffg;
    sb_gain        fio_fbg;

   public :

    sb_first_order ( simulator *      sim,
                     const char *     block_name,
                     float            gain,
                     float            pole ) :
      fio_int ( sim, block_name, 1.0, 0.0 ),
      fio_ffg ( sim, block_name, gain*pole ),
      fio_fbg ( sim, block_name, pole )
      {
       fio_int += fio_ffg;
       fio_int -= fio_fbg;
       fio_fbg += fio_int;
```

```
        }

    void
    operator += ( simulation_block *     new_block )
     {
      fio_ffg += new_block;
     }

    void
    operator -= ( simulation_block *     new_block )
     {
      fio_ffg -= new_block;
     }

    float
    output ( float    sim_time )
     {
      return fio_int.output(sim_time);
        }

    operator simulation_block * ( void )
        {
         return &(fio_int);
        }
};

class sb_pid
 {
  protected :

    sb_sum          pid_sum;
    sb_gain         pid_gain;
    sb_integrator    pid_int;
    sb_differentiator pid_diff;

  public :

    sb_pid ( simulator *      sim,
             const char *     block_name,
             float            kp,
             float            ki,
             float            kd ) :
        pid_sum  ( sim, block_name ),
        pid_gain ( sim, block_name, kp ),
        pid_int  ( sim, block_name, ki, 0.0 ),
        pid_diff ( sim, block_name, kd )
        {
         pid_sum += pid_gain;
         pid_sum += pid_int;
         pid_sum += pid_diff;
        }
```

```
    void
    operator += ( simulation_block *     new_block )
          {
           pid_gain += new_block;
           pid_int  += new_block;
           pid_diff += new_block;
          }

    void
    operator -= ( simulation_block *     new_block )
          {
           pid_gain -= new_block;
           pid_int  -= new_block;
           pid_diff -= new_block;
          }

    float
    output ( float    sim_time )
          {
           return pid_sum.output(sim_time);
          }

    operator simulation_block * ( void )
          {
           return &(pid_sum);
          }
 };

class dc_motor_parameters
 {
  public :

    float   Kb;
    float   Kt;
    float   Ra;
    float   La;
    float   J;
    float   f;
    float   td_start;
    float   td_stop;
    float   td_amp;

    dc_motor_parameters ( void )
     {
      Kb = 0.055;
      Kt = 0.00006;
      Ra = 0.2;
      La = 0.02;
      J  = 0.00005;
      f  = 0.0004;
      td_start = 99999.0;
      td_stop  = 99999.9;
```

```
    td_amp   = 0.0;
   }
};

class sb_dc_motor
 {
  protected :

   sb_sum          input_sum;
   sb_first_order   elec_fio;
   sb_gain         torque_gain;
   sb_pulse        torque_dist;
   sb_sum          torque_sum;
   sb_first_order   mech_fio;
   sb_gain         back_emf_gain;

  public :

   sb_dc_motor ( simulator *          sim,
           const char *         block_name,
           dc_motor_parameters &  p ) :
    input_sum    ( sim, block_name ),
    elec_fio     ( sim, block_name, (1.0 / p.La), (p.Ra / p.La) ),
    torque_gain  ( sim, block_name, p.Kt ),
    torque_dist  ( sim, block_name, p.td_start, p.td_stop, p.td_amp ),
    torque_sum   ( sim, block_name ),
    mech_fio     ( sim, block_name, (1.0 / p.J), (p.f / p.J) ),
    back_emf_gain ( sim, block_name, p.Kb )
    {
     input_sum    -= back_emf_gain;
     elec_fio     += input_sum;
     torque_gain  += elec_fio;
     torque_sum   += torque_gain;
     torque_sum   -= torque_dist;
     mech_fio     += torque_sum;
     back_emf_gain += mech_fio;
    }

   void
   operator += ( simulation_block *     new_block )
    {
     input_sum += new_block;
    }

   void
   operator -= ( simulation_block *     new_block )
    {
     input_sum -= new_block;
    }

   float
   output ( float   sim_time )
```

```
    {
     return mech_fio.output(sim_time);
    }

  operator simulation_block * ( void )
    {
     return mech_fio;
    }
};



#endif
```

```
//FILENAME: DIG.CPP
//BY:     Craig Jensen

# include "dig.h"
# include <math.h>

dig::dig ( void )
 {
 outportb( base_addr + 3,128);  // initialize for output
 outportb( base_addr , ~0);     // zero CS
  }

dig::~dig ( void )
 {
 outportb( base_addr , ~0);     // zero CS
 outportb( base_addr + 2, 2);   // stop stepper controller
 for (int t =0;t<10000;t++)
 outportb( base_addr + 2, 4);   // reset controller
 outportb( base_addr + 2, 0);   // zero stepper controller
 }

void
dig::send_out ( float volt )
 {
  unsigned char byte, limits;
  if ( volt >=  127 )
            {
            byte  = 0x7F;
            }
  else if ( volt <= - 127 )
            {
            byte  = 0xFF;
            }
  else
            {
            byte = (unsigned char) abs(volt);
   if (volt < 0) byte = byte | 128;
            }

  limits = inportb(0x303);
  if (  ((limits & 1) != 1 ) && (volt < 0.0) )
    {
    outportb ( base_addr,~0 );
    return;
    }
  if (  ((limits & 2) != 2 ) && (volt > 0.0) )
    {
    outportb ( base_addr,~0 );
    return;
    }
     outportb ( base_addr,~byte );
}
```

```
//FILENAME:  DIG.H
//BY:        Craig Jensen


# ifndef  DIG_INCLUDED
# define  DIG_INCLUDED


# include <dos.h>

# define base_addr  0x390


class dig
  {

   public :
          dig ( unsigned char req_channel );
  dig ( void );
          ~dig ( void );
          void
          send_out ( float volt );
  };

# endif
```

```
/* File ENGINE.H  Header file for Fuzzy Logic Engine Control */
// Craig Jensen


#ifndef ENGINE_H
#define ENGINE_H

#include "flogic.h"
#include "blocks.h"
#include "step.h"

typedef enum {in_x,in_x_dot};
typedef enum {in_neg_l,in_neg_s,in_ze,in_pos_s,in_pos_l};
typedef enum {out_neg_l,out_neg_m,out_neg_s,out_neg_vs,out_ze,out_pos_vs
                                ,out_pos_s,out_pos_m,out_pos_l};


void init_engine_fuzzy_system (fuzzy_system_rec *fl);

class sb_fuzzy : public simulation_block
 {
                protected :

                        fuzzy_system_rec *     fl;
                        float           fuzzy_time;
                        float           last_time;
                        float           output_val;
                        float           prop_gain;
                        float           dev_gain;
                        float           out_gain;
                        int                                             Sys;

                public :

                        sb_fuzzy ( simulator *          sim,
                                                const char *        new_block_name,
                                                fuzzy_system_rec *    new_fl,
                                                float           update_time,
                                                float           new_prop_gain,
                                                float           new_dev_gain,
                                                float           new_out_gain,
                                                int
        Sys_Num
                                        ) :

                        simulation_block( sim,  new_block_name, 2, 1.0)
                                        {
                                        assert(new_fl != NULL);
                                        output_val = 0.0;
                                        last_time = 0.0;
                                        fl = new_fl;
                                        fuzzy_time = update_time;
```

```
                                prop_gain = new_prop_gain;
                                dev_gain = new_dev_gain;
                                out_gain = new_out_gain;
                                init_engine_fuzzy_system(fl);
                                Sys = Sys_Num;
                                }


        virtual
        void
        update ( float    sim_time )
                {
                 float  input[2];
                 input[0] =prop_gain * inputs[0].value(sim_time);
                          input[1] =dev_gain *  inputs[1].value(sim_time);
                          output_val = out_gain * fuzzy_system(input , * fl,Sys);
                          last_time = sim_time;


                }

        virtual
        boolean
        is_ready  ( void)
         {
         if (total_inputs == 2)  return TRUE;
         return FALSE;
         }

        virtual
        float
        output  ( float sim_time)  {
         return output_val;
         }
};

#endif
```

```
// ENGINEINIT.CPP
// Craig Jensen

#include <alloc.h>
#include "engine.h"

void init_engine_rules (fuzzy_system_rec *fl) {
  const int  no_of_x_rules = 25;
  int i:
  for (i = 0;i < no_of_x_rules;i++) {
          fl->rules[i].inp_index[0] = in_x;
          fl->rules[i].inp_index[1] = in_x_dot;
          }
   /* Regions for x and x_dot: */
    fl->rules[0].inp_fuzzy_set[0] = in_neg_l;
    fl->rules[0].inp_fuzzy_set[1] = in_neg_l;
    fl->rules[0].out_fuzzy_set = out_neg_l;
    fl->rules[1].inp_fuzzy_set[0] = in_neg_l;
    fl->rules[1].inp_fuzzy_set[1] = in_neg_s;
    fl->rules[1].out_fuzzy_set = out_neg_m;
    fl->rules[2].inp_fuzzy_set[0] = in_neg_l;
    fl->rules[2].inp_fuzzy_set[1] = in_ze;
    fl->rules[2].out_fuzzy_set = out_neg_s;
    fl->rules[3].inp_fuzzy_set[0] = in_neg_l;
    fl->rules[3].inp_fuzzy_set[1] = in_pos_s;
    fl->rules[3].out_fuzzy_set = out_neg_vs;
    fl->rules[4].inp_fuzzy_set[0] = in_neg_l;
    fl->rules[4].inp_fuzzy_set[1] = in_pos_l;
    fl->rules[4].out_fuzzy_set = out_ze;

    fl->rules[5].inp_fuzzy_set[0] = in_neg_s;
    fl->rules[5].inp_fuzzy_set[1] = in_neg_l;
    fl->rules[5].out_fuzzy_set = out_neg_m;
    fl->rules[6].inp_fuzzy_set[0] = in_neg_s;
    fl->rules[6].inp_fuzzy_set[1] = in_neg_s;
    fl->rules[6].out_fuzzy_set = out_neg_s;
    fl->rules[7].inp_fuzzy_set[0] = in_neg_s;
    fl->rules[7].inp_fuzzy_set[1] = in_ze;
    fl->rules[7].out_fuzzy_set = out_neg_vs;
    fl->rules[8].inp_fuzzy_set[0] = in_neg_s;
    fl->rules[8].inp_fuzzy_set[1] = in_pos_s;
    fl->rules[8].out_fuzzy_set = out_pos_vs;
    fl->rules[9].inp_fuzzy_set[0] = in_neg_s;
    fl->rules[9].inp_fuzzy_set[1] = in_pos_l;
    fl->rules[9].out_fuzzy_set = out_pos_s;

    fl->rules[10].inp_fuzzy_set[0] = in_ze;
    fl->rules[10].inp_fuzzy_set[1] = in_neg_l;
    fl->rules[10].out_fuzzy_set = out_neg_s;
    fl->rules[11].inp_fuzzy_set[0] = in_ze;
    fl->rules[11].inp_fuzzy_set[1] = in_neg_s;
    fl->rules[11].out_fuzzy_set = out_neg_vs;
```

```
    fl->rules[12].inp_fuzzy_set[0] = in_ze;
    fl->rules[12].inp_fuzzy_set[1] = in_ze;
    fl->rules[12].out_fuzzy_set = out_ze;
    fl->rules[13].inp_fuzzy_set[0] = in_ze;
    fl->rules[13].inp_fuzzy_set[1] = in_pos_s;
    fl->rules[13].out_fuzzy_set = out_pos_vs;
    fl->rules[14].inp_fuzzy_set[0] = in_ze;
    fl->rules[14].inp_fuzzy_set[1] = in_pos_l;
    fl->rules[14].out_fuzzy_set = out_pos_s;

    fl->rules[15].inp_fuzzy_set[0] = in_pos_s;
    fl->rules[15].inp_fuzzy_set[1] = in_neg_l;
    fl->rules[15].out_fuzzy_set = out_neg_s;
    fl->rules[16].inp_fuzzy_set[0] = in_pos_s;
    fl->rules[16].inp_fuzzy_set[1] = in_neg_s;
    fl->rules[16].out_fuzzy_set = out_neg_vs;
    fl->rules[17].inp_fuzzy_set[0] = in_pos_s;
    fl->rules[17].inp_fuzzy_set[1] = in_ze;
    fl->rules[17].out_fuzzy_set = out_pos_vs;
    fl->rules[18].inp_fuzzy_set[0] = in_pos_s;
    fl->rules[18].inp_fuzzy_set[1] = in_pos_s;
    fl->rules[18].out_fuzzy_set = out_pos_s;
    fl->rules[19].inp_fuzzy_set[0] = in_pos_s;
    fl->rules[19].inp_fuzzy_set[1] = in_pos_l;
    fl->rules[19].out_fuzzy_set = out_pos_m;

    fl->rules[20].inp_fuzzy_set[0] = in_pos_l;
    fl->rules[20].inp_fuzzy_set[1] = in_neg_l;
    fl->rules[20].out_fuzzy_set = out_ze;
    fl->rules[21].inp_fuzzy_set[0] = in_pos_l;
    fl->rules[21].inp_fuzzy_set[1] = in_neg_s;
    fl->rules[21].out_fuzzy_set = out_pos_vs;
    fl->rules[22].inp_fuzzy_set[0] = in_pos_l;
    fl->rules[22].inp_fuzzy_set[1] = in_ze;
    fl->rules[22].out_fuzzy_set = out_pos_s;
    fl->rules[23].inp_fuzzy_set[0] = in_pos_l;
    fl->rules[23].inp_fuzzy_set[1] = in_pos_s;
    fl->rules[23].out_fuzzy_set = out_pos_m;
    fl->rules[24].inp_fuzzy_set[0] = in_pos_l;
    fl->rules[24].inp_fuzzy_set[1] = in_pos_l;
    fl->rules[24].out_fuzzy_set = out_pos_l;
    return;
    }

void init_engine_mem_fns (fuzzy_system_rec *fl) {
  /* The X membership functions */
  fl->inp_mem_fns[in_x][in_neg_l] = init_trapz (-1.0,-0.25,0,0,left);
        fl->inp_mem_fns[in_x][in_neg_s] = init_trapz (-1.0,-0.5,-0.5,0,regular);
        fl->inp_mem_fns[in_x][in_ze] = init_trapz (-0.1,0,0,0.1,regular);
        fl->inp_mem_fns[in_x][in_pos_s] = init_trapz (0,0.5,0.5,1.0,regular);
  fl->inp_mem_fns[in_x][in_pos_l] = init_trapz (0.25,1.0,0,0,right);
  /* The X dot membership functions */
```

```
fl->inp_mem_fns[in_x_dot][in_neg_l] = init_trapz (-1.0,-0.5,0,0,left);
        fl->inp_mem_fns[in_x_dot][in_neg_s] = init_trapz (-1.0,-0.5,-0.5,0.0,regular);
        fl->inp_mem_fns[in_x_dot][in_ze] = init_trapz (-0.1,0,0,0.1,regular);
        fl->inp_mem_fns[in_x_dot][in_pos_s] = init_trapz (0,0.5,0.5,1.0,regular);
        fl->inp_mem_fns[in_x_dot][in_pos_l] = init_trapz (0.5,1.0,0,0,right);
return;
}

void init_engine_fuzzy_system (fuzzy_system_rec *fl) {
        fl->no_of_inputs = 2;   /* Inputs are handled 2 at a time only */
        fl->no_of_rules = 25;
fl->no_of_inp_regions = 5;
fl->no_of_outputs = 9;
fl->output_values [out_neg_l] = -1.0;
fl->output_values [out_neg_m] = -.55;
fl->output_values [out_neg_s] = -0.3;
fl->output_values [out_neg_vs] = -0.1;
fl->output_values [out_ze] = 0.0;
fl->output_values [out_pos_vs] = .1;
fl->output_values [out_pos_s] = 0.3;
fl->output_values [out_pos_m] = .55;
fl->output_values [out_pos_l] = 1.0;
fl->rules = (rule *) malloc ((size_t)(fl->no_of_rules*sizeof(rule)));
init_engine_rules(fl);
init_engine_mem_fns(fl);
return;
}
```

```
/* File filt.H  Header file for Fuzzy Logic engine control */
// Craig Jensen

#ifndef filt_H
#define filt_H

#include "flogic.h"
#include "blocks.h"
#include "step.h"

typedef enum {in_xx,in_xx_dot};
typedef enum {in_neg,in_pos};
typedef enum {out_neg,out_zer,out_pos};

void init_filt_fuzzy_system (fuzzy_system_rec *fl);

class sb_adv_fuzzy : public simulation_block
 {
                protected :
                        fuzzy_system_rec *     fl;
                        float                  fuzzy_time;
                        float                  last_time;
                        float                  output_val;
                        float                  prop_gain;
                        float                  dev_gain;
                        float                  out_gain;
                        int                    Sys;
                        float                  last_err,this_err;
                public :
                sb_adv_fuzzy ( simulator *          sim,
                                                const char *        new_block_name,
                                                fuzzy_system_rec *  new_fl,
                                                float               update_time,
                                                float               new_prop_gain,
                                                float               new_dev_gain,
                                                float               new_out_gain,
                                                int                 Sys_Num
                                        ) :
                        simulation_block( sim,  new_block_name, 2, 1.0)
                                        {
                                        assert(new_fl != NULL);
                                        output_val = 0.0;
                                        last_time = 0.0;
                                        fl = new_fl;
                                        fuzzy_time = update_time;
                                        prop_gain = new_prop_gain;
                                        dev_gain = new_dev_gain;
                                        out_gain = new_out_gain;
                                        init_filt_fuzzy_system(fl);
                                        Sys = Sys_Num;
                                       this_err = 0;
                                       last_err  = 0;
```

```
    }

            virtual
            void
            update ( float    sim_time )
                    {
                     float  input[2];
                     input[0] =prop_gain * inputs[0].value(sim_time);
    input[1] =dev_gain *  inputs[1].value(sim_time);
    this_err = inputs[0].value(sim_time) / 0.005;
    if ( (this_err * last_err) < 0.0) //reset gain on zero cross
       {
       prop_gain = 1.0;
       out_gain = 1.0;
       }

    else if ( xfabs(this_err) > xfabs(last_err) )  // set gain on max error
       {
       prop_gain = 5.0/150.0*(200.0-xfabs(this_err))+1.0;
       if (prop_gain > 5.0) prop_gain = 5.0;
       if (prop_gain < 1.0) prop_gain = 1.0;
       out_gain = 3.0/150.0*(200.0-xfabs(this_err))+1;
       if (out_gain > 3.0) out_gain = 3.0;
       if (out_gain < 1.0) out_gain = 1.0;
       }

    output_val = out_gain * fuzzy_system(input , * fl,Sys);
                                last_time = sim_time;
    last_crr = this_err;

                    }

            virtual
            boolean
            is_ready  ( void)
             {
             if (total_inputs == 2)  return TRUE;
             return FALSE;
             }

            virtual
            float
            output  ( float sim_time)  {
             return output_val;
             }
 };
#endif
```

```
// FILTINIT.CPP Initialization procedures for Fuzzy Logic Overshoot Filter Control */
// Craig Jensen

#include <alloc.h>
#include "filt.h"

void init_filt_rules (fuzzy_system_rec *fl) {
  const int  no_of_x_rules = 4;
  int i;
  for (i = 0;i < no_of_x_rules;i++) {
            fl->rules[i].inp_index[0] = in_xx;
            fl->rules[i].inp_index[1] = in_xx_dot;
            }
   /* Regions for x and x_dot: */
    fl->rules[0].inp_fuzzy_set[0] = in_neg;
    fl->rules[0].inp_fuzzy_set[1] = in_neg;
    fl->rules[0].out_fuzzy_set = out_zer;
    fl->rules[1].inp_fuzzy_set[0] = in_neg;
    fl->rules[1].inp_fuzzy_set[1] = in_pos;
    fl->rules[1].out_fuzzy_set = out_neg;
    fl->rules[2].inp_fuzzy_set[0] = in_pos;
    fl->rules[2].inp_fuzzy_set[1] = in_neg;
    fl->rules[2].out_fuzzy_set = out_pos;
    fl->rules[3].inp_fuzzy_set[0] = in_pos;
    fl->rules[3].inp_fuzzy_set[1] = in_pos;
    fl->rules[3].out_fuzzy_set = out_zer;
    return;
    }
void init_filt_mem_fns (fuzzy_system_rec *fl) {
  /* The X membership functions */
  fl->inp_mem_fns[in_xx][in_neg] = init_trapz (-1,-0.0001,-0.0001,0,regular);
  fl->inp_mem_fns[in_xx][in_pos] = init_trapz (0,0.0001,0.0001,1,regular);
   /* The X dot membership functions */
  fl->inp_mem_fns[in_xx_dot][in_neg] = init_trapz (-1.0,0,0,0,left);
            fl->inp_mem_fns[in_xx_dot][in_pos] = init_trapz (0,1.0,0,0,right);
            return;   }
void init_filt_fuzzy_system (fuzzy_system_rec *fl) {
           fl->no_of_inputs = 2;   /* Inputs are handled 2 at a time only */
           fl->no_of_rules = 4;
  fl->no_of_inp_regions = 2;
  fl->no_of_outputs = 3;
  fl->output_values [out_neg] = -1.0;
  fl->output_values [out_zer] =  0.0;
  fl->output_values [out_pos] = 1.0;
  fl->rules = (rule *) malloc ((size_t)(fl->no_of_rules*sizeof(rule)));
  init_filt_rules(fl);
  init_filt_mem_fns(fl);     return;
  }
```

```
/*  File FLFILES.CPP  Routines to read and write fuzzy system files  */

#ifndef FLFILES_C
#define FLFILES_C
#include <conio.h>
#include <alloc.h>
#include <stdio.h>
#include <windows.h>
#include "ut.h"
#include "flogic.h"

short read_fuzzy_system (path_str the_file_name,
        fuzzy_system_rec *fz) {
#define MAX_REC_LEN 255
        int i,j;
        long file_size;
        char char_str[MAX_REC_LEN];
        FILE *the_file;
        //clrscr();
        gotoxy (1,5);
        file_size = open_input_text_file (&the_file,the_file_name);
        if (file_size == 0) return 0;  /* File name not found  */
        /*  Check system parameters */
        fgets (char_str,MAX_REC_LEN,the_file);
        sscanf (char_str,"%d %d %d %d",&(fz->no_of_inputs),
       &(fz->no_of_inp_regions),
       &(fz->no_of_rules),&(fz->no_of_outputs));
        /* Read output values */
        for (i=0;i<fz->no_of_outputs;i++) {
       fgets (char_str,MAX_REC_LEN,the_file);
       sscanf (char_str,"%f",&(fz->output_values[i]));
       }  /* end i  */
        /* Read membership functions */
        for (i=0;i<fz->no_of_inputs;i++)
       for (j=0;j<fz->no_of_inp_regions;j++) {
                fgets(char_str,MAX_REC_LEN,the_file);
                sscanf(char_str,"%f %f %f %f %f %f",
               &(fz->inp_mem_fns[i][j].a),&(fz->inp_mem_fns[i][j].b),
               &(fz->inp_mem_fns[i][j].c),&(fz->inp_mem_fns[i][j].d),
               &(fz->inp_mem_fns[i][j].l_slope),
               &(fz->inp_mem_fns[i][j].r_slope));
                } /*  end i,j  */
        /*  Allocate space for rules */
        fz->rules = (rule *) malloc ((size_t)(fz->no_of_rules*sizeof(rule)));
        /* Read rules.  Length of record varies with number of inputs. */
        for (i=0;i<fz->no_of_rules;i++) {
       for (j=0;j<fz->no_of_inputs;j++)
                fscanf (the_file,"%hd %hd",&(fz->rules[i].inp_index[j]),
               &(fz->rules[i].inp_fuzzy_set[j]));
       fscanf (the_file,"%hd\n",&(fz->rules[i].out_fuzzy_set));
       }  /* end i */
```

```
        fclose (the_file);
        char buffer[100];
        sprintf (buffer,"Done reading fuzzy system file %s \0",the_file_name);
        MessageBox(NULL,buffer,"Open",MB_OK);
        return 1;
        } /* end func */

short write_fuzzy_system (path_str the_file_name,fuzzy_system_rec fz)
        {
        int i,j;
        FILE *the_file;
        char buffer[100];
        gotoxy (1,5);
        if (!open_output_text_file (&the_file,the_file_name)) return 0;
        /* Write system parameters */
        fprintf (the_file,"%d %d %d %d\n",fz.no_of_inputs,
        fz.no_of_inp_regions,
        fz.no_of_rules,fz.no_of_outputs);
        /* Write output values */
        for (i=0;i<fz.no_of_outputs;i++)
                {
                fprintf (the_file,"%5.2f\n",fz.output_values[i]);
                } /* end i */
        /* Write membership functions */
        for (i=0;i<fz.no_of_inputs;i++)
                for (j=0;j<fz.no_of_inp_regions;j++)
                        {
                        fprintf(the_file,"%6.4f %6.4f %6.4f %6.4f %6.4f %6.4f\n",
                                fz.inp_mem_fns[i][j].a,fz.inp_mem_fns[i][j].b,
                                fz.inp_mem_fns[i][j].c,fz.inp_mem_fns[i][j].d,
                                fz.inp_mem_fns[i][j].l_slope,
                                fz.inp_mem_fns[i][j].r_slope);
                        } /* end i,j */
        /* Write rules. Length of record varies with number of inputs. */
        for (i=0;i<fz.no_of_rules;i++)
                {
                for (j=0;j<fz.no_of_inputs;j++)
                        fprintf (the_file,"%hd %hd ",fz.rules[i].inp_index[j],
                                fz.rules[i].inp_fuzzy_set[j]);
                fprintf (the_file," %hd\n",fz.rules[i].out_fuzzy_set);
                } /* end i */
        fclose (the_file);
        sprintf (buffer,"Done writing fuzzy system file %s \0",the_file_name);
        MessageBox(NULL,buffer,"Save",MB_OK);
        return 1;
        } /* end func */

#endif
```

```
/* File FLOGIC.H  Fuzzy logic header file  */

#ifndef FLOGIC_H
#define FLOGIC_H
#include "uttypes.h"
#define MAX_NO_OF_INPUTS 5
#define MAX_NO_OF_INP_REGIONS 5
#define MAX_NO_OF_OUTPUT_VALUES 10

typedef enum {regular,left,right} trapz_type;

typedef struct {
        float weight_x;
        float weight_y;
        int   num_rules_fired;
        } f_out_record;

extern f_out_record out_record[25];

typedef struct {
        trapz_type tp;
        float a,b,c,d,l_slope,r_slope;
        }  trapezoid;

typedef struct {
        short inp_index[MAX_NO_OF_INPUTS],
         inp_fuzzy_set[MAX_NO_OF_INPUTS],
         out_fuzzy_set;
         }  rule;

typedef struct {
        short allocated;
        trapezoid inp_mem_fns [MAX_NO_OF_INPUTS] [MAX_NO_OF_INP_REGIONS];
        far rule *rules;
        int no_of_inputs,no_of_inp_regions,no_of_rules,no_of_outputs;
        float output_values[MAX_NO_OF_OUTPUT_VALUES];
        } fuzzy_system_rec;

extern fuzzy_system_rec g_fuzzy_system;
extern fuzzy_system_rec f_fuzzy_system;

/*  File FLPRCS.CPP */
trapezoid init_trapz (float x1,float x2,float x3,float x4,trapz_type typ);
float fuzzy_system (float inputs[],fuzzy_system_rec fl,int Sys_Num);
void free_fuzzy_rules (fuzzy_system_rec *fz);

/*  File FLFILES.CPP  */
short read_fuzzy_system (path_str the_file_name,  fuzzy_system_rec *fz);
short write_fuzzy_system (path_str the_file_name,fuzzy_system_rec fz);

#endif
```

```
/*  File FLPRCS.CPP  Fuzzy Logic procedures and functions  */

#ifndef FLPRCS_CPP
#define FLPRCS_CPP

#define TOO_SMALL 1e-6

#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "simulate.h"
#include "flogic.h"

fuzzy_system_rec g_fuzzy_system;
fuzzy_system_rec f_fuzzy_system;
f_out_record  out_record[25];

trapezoid init_trapz (float x1,float x2,float x3,float x4,
        trapz_type typ);
float fuzzy_system (float inputs[],fuzzy_system_rec fl, int Sys_Num);
void free_fuzzy_rules (fuzzy_system_rec *fz);

/*  Implementation  */

trapezoid init_trapz (float x1,float x2,float x3,float x4,
        trapz_type typ) {
        trapezoid trz;
        trz.a = x1;
        trz.b = x2;
        trz.c = x3;
        trz.d = x4;
        trz.tp = typ;
        switch (trz.tp) {
                case regular:
         trz.l_slope = 1.0/(trz.b - trz.a);
         trz.r_slope = 1.0/(trz.c - trz.d);
         break;
                case left:
         trz.r_slope = 1.0/(trz.a - trz.b);
         trz.l_slope = 0.0;
         break;
                case right:
         trz.l_slope = 1.0/(trz.b - trz.a);
         trz.r_slope = 0.0;
         break;
                } /* end switch  */
        return trz;
        } /* end function */

float trapz (float x, trapezoid trz) {
        switch (trz.tp) {
                case left:
```

```
        if (x <= trz.a)
                return 1.0;
        if (x >= trz.b)
                return 0.0;
        /* a < x < b */
        return trz.r_slope * (x - trz.b);
                case right:
        if (x <= trz.a)
                return 0.0;
        if (x >= trz.b)
                return 1.0;
        /* a < x < b */
        return trz.l_slope * (x - trz.a);
                case regular:
        if ((x <= trz.a) || (x >= trz.d))
                return 0.0;
        if ((x >= trz.b) && (x <= trz.c))
                return 1.0;
        if ((x >= trz.a) && (x <= trz.b))
                return trz.l_slope * (x - trz.a);
        if ((x >= trz.c) && (x <= trz.d))
                return  trz.r_slope * (x - trz.d);
        } /* End switch */
  return 0.0; /* should not get to this point */
  } /* End function */

float min_of (float values[],int no_of_inps) {
  int i;
  float val;
  val = values [0];
  for (i = 1;i < no_of_inps;i++) {
                if (values[i] < val)
          val = values [i];
        }
        return val;
        }

float fuzzy_system (float inputs[],fuzzy_system_rec fz,int Sys_Num) {
        int i,j,k=0;
        short variable_index,fuzzy_set;
        float sum1 = 0 0,sum2 = 0.0,weight;
        float m_values[MAX_NO_OF_INPUTS];
        if (Sys_Num == 1) out_record[0].num_rules_fired = 0;
        for (i = 0;i < fz.no_of_rules;i++) {
                for (j = 0;j < fz.no_of_inputs;j++) {
                variable_index = fz.rules[i].inp_index[j];
                fuzzy_set = fz.rules[i].inp_fuzzy_set[j];
                m_values[j] = trapz(inputs[variable_index],
                        fz.inp_mem_fns[variable_index][fuzzy_set]);
                } /* end j */
                weight = min_of (m_values,fz.no_of_inputs);
                if ((weight > 0) && (Sys_Num == 1))
```

```
                        {
                        out_record[k].weight_y = weight;
                        out_record[k].weight_x = fz.output_values[fz.rules[i].out_fuzzy_set];
                        out_record[0].num_rules_fired++;
                        k++;
                        }
                sum1 += weight * fz.output_values[fz.rules[i].out_fuzzy_set];
                sum2 += weight;
                } /* end i  */
        if (fabs(sum2) < TOO_SMALL) {
                //cprintf ("\r\nFLPRCS Error: Sum2 in fuzzy_system is 0.  Press key: ");
                //getch();
                //exit (1);
                return 0;
                }
 if (Sys_Num == 2) return sum1;
        return (sum1/sum2);
        }  /* end fuzzy_system  */

void free_fuzzy_rules (fuzzy_system_rec *fz) {
         if (fz->allocated)
                 free (fz->rules);
         fz->allocated = FALSE;
         return;
   }

#endif
```

```
/*------------------------
        MENU.H header file
 ------------------------*/
// Craig Jensen


#define IDM_OPEN        1
#define IDM_SAVE        2
#define IDM_SAVEAS      3
#define IDM_EXIT        4

#define IDM_STEP        10
#define IDM_ERROR       11
#define IDM_DERIVATIVE  12
#define IDM_OUTRULES    13
#define IDM_FILTER      14
#define IDM_TILE        15
#define IDM_START       16
#define IDM_STOP        17

#define IDM_STARTSIM      20
#define IDM_STOPIMP       21
#define IDM_PAUSESIM      22
#define IDM_RESETSIM      23
#define IDM_STARTIMP      24
#define IDM_PAUSEIMP      25
#define IDM_S_TIME        26


#define IDM_CLOSE       30

#define IDM_GAIN        40

#define IDM_HELP        50
#define IDM_ABOUT       51
```

```
// Model.h
// Craig Jensen  1995

#include "engine.h"
#include "filt.h"

const float  System_Delta_Time = 0.0010;
const float  System_Run_Time  = 2.0;
const float  step_size       = 3600.0;
const int    MAX_DATA         = 300;
const float  eng_delay       = 0.025;
const float  ld1_on_time      = 0.1;
const float  ld1_off_time     = 5.0;
const float  ld1_mag         = 3.0;
const float  ld2_on_time      = 2.0;
const float  ld2_off_time     = 6.0;
const float  ld2_mag         = 0.0;
const float  ld3_on_time      = 3.0;
const float  ld3_off_time     = 7.0;
const float  ld3_mag         = 0.0;
const float  g5             = 0.003;
const float  g6             = 0.001;
const float  g7             = 220.0;
const float  max_x           = 150.0;
const float  min_x           = -100.0;
const float  par_var         = 1.0;
const float  k2             = 10*par_var;
const float  k3             = 0.001007*par_var;
const float  k4             = 0.23*par_var;
const float  k6             = 0.0011*par_var;
const float  j              = 0.003212*par_var;

class system_model
        {
        public :

                simulator       sim;

                sb_pulse        ref;
                sb_pulse        out;
                sb_sum          sum1;
                sb_sum          sum2;
                sb_sum          sum3;
                sb_delay        del1;
                sb_fuzzy        fuzzy;
                sb_adv_fuzzy        advfuzzy;
                sb_lim_integrator int2;
                sb_differentiator dif;
                sb_first_order    fio;
                sb_gain             gai1;
                sb_delay        del2;
                sb_integrator     int1;
```

```
sb_gain         gai2;
sb_gain         gai3;
sb_gain         gai4;
sb_gain         gai5;
sb_gain         gai6;
sb_gain         gai7;
sb_gain         gai8;
sb_gain         gai9;
sb_gain         gai10;
sb_pulse        load1;
sb_pulse        load2;
sb_pulse        load3;

sb_sum          nsum1;
sb_sum          nsum2;
sb_fuzzy        nfuzzy;
sb_lim_integrator    nint2;
sb_differentiator ndif;
sb_first_order    nfio;
sb_gain         ngai1;
sb_integrator     nint1;
sb_gain         ngai2;
sb_gain         ngai3;
sb_gain         ngai4;
sb_gain         ngai5;
sb_gain         ngai6;
sb_gain         ngai7;
sb_delay                        ndel1;
sb_delay                        ndel2;

sb_sum          psum1;
sb_sum          psum2;
sb_pid          pid;
sb_lim_integrator    pint2;
sb_first_order    pfio;
sb_gain         pgai1;
sb_integrator     pint1;
sb_gain         pgai2;
sb_gain         pgai3;
sb_gain         pgai4;
sb_delay                        pdel1;
sb_delay                        pdel2;


system_model ( ):
        sim      ( System_Run_Time, System_Delta_Time ),

        ref   (sim,"ref_input", 0,System_Run_Time  , step_size ),
        out   (sim,"ref_speed", 0,System_Run_Time, 3600 ),
        sum1   (sim,"sum1" ),
        sum2   (sim,"sum2" ),
        sum3   (sim,"sum3" ),
```

```
del1  (sim,"del1",.002,1000,1.0),
fuzzy (sim,"fuzzy", &g_fuzzy_system,System_Delta_Time,1,1,1,1),
advfuzzy (sim,"advfuzzy",&f_fuzzy_system,System_Delta_Time,1,1,1,2),
int2  (sim,"int2",1.0,0,-15.6,84.4),
dif   (sim,"dif",1.0),
fio   (sim,"fio",1,k2),
gai1  (sim,"gai1",k4),
del2  (sim,"del2",eng_delay,1000,1.0),
int1  (sim,"int",1.0,0.0),
gai2  (sim,"gai2",1.0/j),
gai3  (sim,"gai3",k6),
gai4  (sim,"gai4",k3),
gai5  (sim,"gai5",g5),
gai6  (sim,"gai6",g6),
gai7  (sim,"gai7",g7),
gai8  (sim,"gai8",1.0),
gai9  (sim,"gai9",1.0),
gai10 (sim,"gai10",1.0),
load1  (sim,"load1", ld1_on_time, ld1_off_time, ld1_mag ),
load2  (sim,"load2", ld2_on_time, ld2_off_time, ld2_mag ),
load3  (sim,"load3", ld3_on_time, ld3_off_time, ld3_mag ),

nsum1  (sim,"nsum1" ),
nsum2  (sim,"nsum2" ),
ndel1  (sim,"del1",.002,1000,1.0),
nfuzzy (sim,"fuzzy", &g_fuzzy_system,System_Delta_Time,1,1,1,3),
nint2  (sim,"int2",1.0,0,-15.6,84.4),
ndif   (sim,"dif",1.0),
nfio   (sim,"fio",1,k2),
ngai1  (sim,"gai1",k4),
ndel2  (sim,"del2",eng_delay,1000,1.0),
nint1  (sim,"int",1.0,0.0),
ngai2  (sim,"gai2",1.0/j),
ngai3  (sim,"gai3",k6),
ngai4  (sim,"gai4",k3),
ngai5  (sim,"ngai5",g5),
ngai6  (sim,"ngai6",g6),
ngai7  (sim,"ngai7",g7),

psum1  (sim,"psum1" ),
psum2  (sim,"psum2" ),
pdel1  (sim,"del1",.002,1000,1.0),
pid    (sim,"pid", 0.2,0,0.07),
pint2  (sim,"int2",1.0,0,-15.6,84.4),
pfio   (sim,"fio",1,k2),
pgai1  (sim,"gai1",k4),
pdel2  (sim,"del2",eng_delay,1000,1.0),
pint1  (sim,"int",1.0,0.0),
pgai2  (sim,"gai2",1.0/j),
pgai3  (sim,"gai3",k6),
pgai4  (sim,"gai4",k3)
```

```
{
// Connect The Fuzzy Simulation Model...
sum3  += load1;
sum3  += load2;
sum3  += load3;
sum1  += ref;
sum1  -= sum2;
del1  += sum1;
gai5  += del1;
gai6  += dif;
dif   += del1;
fuzzy  += gai5;
fuzzy  += gai6;
advfuzzy  += gai8;
advfuzzy  += gai9;
gai8     += del1;
gai9     += dif;
gai10    += advfuzzy;
gai7  += fuzzy;
int2  += gai7;
int2  -= gai10;
fio   += int2;
fio   -= gai4;
gai1  += fio;
del2  += gai1;
int1  += del2;
int1  -= sum3;
int1  -= gai3;
gai2  += int1;
gai3  += gai2;
gai4  += gai2;
sum2  += out;
sum2  += gai2;

nsum1  += ref;
nsum1  -= nsum2;
ndif   += ndel1;
ndel1  += nsum1;
ngai5  += ndel1;
ngai6  += ndif;
nfuzzy  += ngai5;
nfuzzy  += ngai6;
ngai7  += nfuzzy;
nint2  += ngai7;
nfio   += nint2;
nfio   -= ngai4;
ngai1  += nfio;
ndel2  += ngai1;
nint1  += ndel2;
nint1  -= sum3;
```

```
        nint1  -= ngai3;
        ngai2  += nint1;
        ngai3  += ngai2;
        ngai4  += ngai2;
        nsum2  += out;
        nsum2  += ngai2;

        psum1  += ref;
        psum1  -= psum2;
        pdel1  += psum1;
        pid    += pdel1;
        pint2  += pid;
        pfio   += pint2;
        pfio   -= pgai4;
        pgai1  += pfio;
        pdel2  += pgai1;
        pint1  += pdel2;
        pint1  -= sum3;
        pint1  -= pgai3;
        pgai2  += pint1;
        pgai3  += pgai2;
        pgai4  += pgai2;
        psum2  += out;
        psum2  += pgai2;

        }
~system_model(void)
        {

        free_fuzzy_rules(&g_fuzzy_system);
        free_fuzzy_rules(&f_fuzzy_system);
        }
};
```

```
// SIMULATE.CPP
//  Brad Trostad and Craig Jensen

#include <assert.h>
#include <stdio.h>
#include <string.h>
#include "simulate.h"


// 'input_node' Class Public Source Code Starts Here...
input_node::
input_node ( void )
 {
  block = NULL;
  sign  = POSITIVE;
 }

boolean
input_node::
is_assigned ( void )
 {
  if ( block != NULL )
    {
                    return TRUE;
    }
  return FALSE;
 }

void
input_node::
operator += ( simulation_block *   block_ptr )
 {
  assert ( block_ptr != NULL );
  block = block_ptr;
  sign  = POSITIVE;
 }

void
input_node::
operator -= ( simulation_block *   block_ptr )
 {
         assert ( block_ptr != NULL );
  block = block_ptr;
  sign  = NEGATIVE;
 }


float
input_node ::
value(float sim_time) {
            return block->output(sim_time)*float(sign);
            }
```

```
// 'simulation_block' Class Protected Source Code Starts Here...
float
simulation_block::
input_sum ( float     sim_time )
 {
  float   sum = 0.0;
          int     i;

  for ( i = 0; i < total_inputs; ++i )
   {
    sum += inputs[i].value(sim_time);
   }
  return sum;
 }

// 'simulation_block' Class Public Source Code Starts Here...
simulation_block::
simulation_block ( simulator *     sim,
              const char *    new_block_name,
              int             requested_inputs,
              float           output_gain_value )
 {
  assert ( sim != NULL );
  strcpy ( block_name, new_block_name );
          inputs      = NULL;
  total_inputs = 0;
  max_inputs   = 0;
  output_gain  = output_gain_value;
  if ( requested_inputs > 0 )
   {
    inputs = new input_node [requested_inputs];
    assert ( inputs != NULL );
    max_inputs = requested_inputs;
   }
  sim->add_block ( this );
 }

simulation_block::
~simulation_block ( void )
 {
  if ( max_inputs > 0 )
   {
                    delete inputs;
   }
 }

void
simulation_block::
operator += ( simulation_block *   block_ptr )
 {
  assert ( total_inputs < max_inputs );
```

```
  inputs[total_inputs] += block_ptr;
  ++total_inputs;
 }

void
simulation_block::
operator -= ( simulation_block *   block_ptr )
 {
  assert ( total_inputs < max_inputs );
        inputs[total_inputs] -= block_ptr;
  ++total_inputs;
 }

// 'latched_simulation_block' Class Public Source Code Starts Here...
latched_simulation_block::
latched_simulation_block ( simulator *     sim,
                const char *    new_block_name,
                int             requested_inputs,
                float           output_gain_value ) :
 simulation_block(sim,new_block_name,requested_inputs,output_gain_value)
 {
  output_value = 0.0;
  latch_value  = 0.0;
 }

float
latched_simulation_block::
output ( float    sim_time )
 {
        return (output_value * output_gain);
 }

void
latched_simulation_block::
latch ( void )
 {
        output_value = latch_value;
 }

// 'simulator' Class Public Source Code Starts Here...
simulator::
simulator ( float      max_time,
                                float      time_step )
 {
        int  i;

        sim_max_time  = max_time;
  sim_curr_time  = 0.0;
  sim_time_step  = time_step;
  sim_iterations = 0;
  blocks_used = 0;
  for ( i = 0; i < MAX_SIMULATION_BLOCKS; ++i )
```

```
    {
     blocks[i] = NULL;
    }
 }

simulator::
~simulator ( void )
 {
  // Does Nothing So Far...
 }

void
simulator::
add_block ( simulation_block *    new_block )
 {
  assert ( new_block != NULL );
  assert ( blocks_used < MAX_SIMULATION_BLOCKS );
  blocks[blocks_used] = new_block;
  ++blocks_used;
 }

boolean
simulator::
is_ready ( void )
 {
  int    i;

  for ( i = 0; i < blocks_used; ++i )
    {
     if ( !blocks[i]->is_ready() )
       {
        return FALSE;
       }
    }
  return TRUE;
 }

boolean
simulator::
has_finished ( void )
 {
  if ( elapsed_time() > sim_max_time )
    {
     return TRUE;
    }
  return FALSE;
 }

boolean
simulator::
update ( void )
 {
```

```
 return update(sim_iterations * sim_time_step);
 }

boolean
simulator::
update ( float new_elapsed_time )
 {
  int   i;

  if ( sim_iterations == 0 )
   {
    assert ( is_ready() );
   }
  if ( has_finished() )
   {
    return FALSE;
   }
  for ( i = 0; i < blocks_used; ++i )
   {
    blocks[i]->update ( new_elapsed_time );
   }
  for ( i = 0; i < blocks_used; ++i )
   {
    blocks[i]->latch();
   }
  sim_curr_time = new_elapsed_time;
  ++sim_iterations;
  return TRUE;
 }
```

```
// SIMULATE.H
// Brad Trostad and Craig Jensen


#ifndef SIMULATE_INCLUDED
#define SIMULATE_INCLUDED

#define MAX_BLOCK_INPUTS        10
#define MAX_SIMULATION_BLOCKS   100

#define POSITIVE   1
#define NEGATIVE  -1
#define TRUE    1
#define FALSE   0

typedef int          boolean;
typedef unsigned long  dword;

class simulation_block;
class simulator;

class input_node
 {
  public :

    simulation_block *    block;
    int                   sign;

    input_node  ( void );

    boolean
    is_assigned ( void );

    void
    operator += ( simulation_block *   block_ptr );

    void
    operator -= ( simulation_block *   block_ptr );

    float
    value(float sim_time) ;

 };


class simulation_block
 {
  protected :

    char         block_name[30];
    input_node *  inputs;
    int          max_inputs;
```

```
    int         total_inputs;

                float
                input_sum ( float   sim_time );

        public :

                float       output_gain;
                simulation_block ( simulator *     sim,
                                                        const char *    new_block_name,
                                                        int             requested_inputs,
                                                        float           output_gain_value );

                virtual
         ~simulation_block ( void );

                void
                operator += ( simulation_block *   block_ptr );

                void
                operator -= ( simulation_block *   block_ptr );

                virtual
                boolean
                is_ready ( void ) = 0;

                virtual
                float
                output ( float     sim_time ) = 0;

                virtual
                void
                update ( float     sim_time )
                 {
                        // Does Nothing At This Level...
                 }

                virtual
                void
                latch ( void )
                 {
                        // Does Nothing At This Level...
                 }

                operator simulation_block * ( void )
                 {
                        return this;
    }
 };

class latched_simulation_block : public simulation_block
 {
  protected :
```

```
    float    output_value;
    float    latch_value;

  public :

    latched_simulation_block ( simulator *     sim,
                       const char *    new_block_name,
                       int             requested_inputs,
                       float           output_gain_value );
    virtual
    float
    output ( float    sim_time );

    virtual
    void
    latch ( void );
 };

class simulator
 {
  protected :

    simulation_block *   blocks[MAX_SIMULATION_BLOCKS];
    int              blocks_used;
    float            sim_max_time;
    float            sim_curr_time;
    float            sim_time_step;
    dword              sim_iterations;


  public :

    simulator ( float      max_time,
            float      time_step );

   ~simulator ( void );

    void
    add_block ( simulation_block *    new_block );

    boolean
    is_ready ( void );

    boolean
    has_finished ( void );

    dword
    iterations ( void )
     {
      return sim_iterations;
     }
```

```
    float
    elapsed_time ( void )
      {
       return (sim_curr_time);
      }

    float
    time_remaining ( void )
      {
       return (sim_max_time - elapsed_time() );
      }

    boolean
    update ( void );

    boolean
    update ( float new_elapsed_time );

    operator simulator * ( void )
      {
       return this;
      }
};

#endif
```

```
/*-------------------------------------------------------
        STEP.CPP -- Runs step response with Fuzzy controller
                                        Craig Jensen,  1995
 -------------------------------------------------------*/

#include <windows.h>
#include <time.h>
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <string.h>
#include "engine.h"
#include "filt.h"
#include "menu.h"
#include "model.h"
#include "step.h"

#define UP 1
#define DOWN 0
#define IDC_SMALLER 1
#define IDC_LARGER 2
#define IDC_OUTSMALLER 3
#define IDC_OUTLARGER 4
#define BTN_HEIGHT 20
#define BTN_WIDTH  20
int Cycles_per_Timer = 100 ;
const int num_to_ave = 10;


long FAR PASCAL _export WndProc (HWND, UINT, UINT, LONG) ;
long FAR PASCAL _export WPStep (HWND, UINT, UINT, LONG) ;
long FAR PASCAL _export FuzErrMemProc (HWND, UINT, UINT, LONG) ;
long FAR PASCAL _export FuzDerMemProc (HWND, UINT, UINT, LONG) ;
long FAR PASCAL _export FuzOutMemProc (HWND, UINT, UINT, LONG) ;
long FAR PASCAL _export FuzFilt1MemProc (HWND, UINT, UINT, LONG) ;
long FAR PASCAL _export FuzFilt2MemProc (HWND, UINT, UINT, LONG) ;


char szTimeResp[] = "Time Response" ;
char szFuzErr[] = "Error Memberships" ;
char szFuzDer[] = "Derivative Memberships" ;
char szFuzOut[] = "Fuzzy Output" ;
char szFilter1[] = "Adv. Derivative Memberships" ;
char szFilter2[] = "Adv. Error Memberships" ;

char file_name[] = "c:\\bc4\\bin\\honda\\fuzzy.dat";

system_model * em       =       NULL;
HMENU                           hMainMenu,hStepMenu,hErrMenu,hDerMenu,hOutMenu;
int                             k=0;
HPEN                            hPen1,hPen2,hPen3,hPen4,hPen5;
float   x0,et=0,et0=0,Err_Gain,Der_Gain,Out_Gain,ROA,e1,e0,ROA_GAIN=0.0003;
```

```
float           ERR_GAIN=0.005,FUZGAIN=50.0;
float           REF=3600,s_time=.0021,input[2],output_val,wrap_time=0,speed;
HANDLE          hInst;
BOOL            TimeOpen,ErrOpen,DerOpen,OutOpen;
BOOL            FilterOpen,SIMULATION,IMPLEMENTATION;
fuzzy_system_rec  fz,fzz;
float ROA_PT =1;
float ERR_PT =1;


float *     data_et;
float *     data1;
float *     data2;
float *     data3;
float *     data4;


int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                                        LPSTR lpszCmdParam, int nCmdShow)
     {
     static char                szAppName[] = "Fuzzy Simulator";
     HWND                       hwnd ;
     MSG                        msg ;
     WNDCLASS                   wndclass ;

    em = new system_model;
    assert(em != NULL);
    fz = g_fuzzy_system;
    fzz = f_fuzzy_system;
    hInst = hInstance;
    // Create colored pens
    hPen1 = CreatePen(PS_SOLID,1,RGB(255,0,0));
    hPen2 = CreatePen(PS_SOLID,1,RGB(0,0,255));
    hPen3 = CreatePen(PS_SOLID,1,RGB(0,255,0));
    hPen4 = CreatePen(PS_SOLID,2,RGB(0,0,0));
    hPen5 = CreatePen(PS_DOT,1,RGB(0,0,0));

    // Allocate memory for data output
    data_et = new float [MAX_DATA];
    data1 = new float [MAX_DATA];
    data2 = new float [MAX_DATA];
    data3 = new float [MAX_DATA];
    data4 = new float [MAX_DATA];
    // Check for NULL pointers
    assert (data_et != NULL);
    assert (data1 != NULL);
    assert (data2 != NULL);
    assert (data3 != NULL);
    assert (data4 != NULL);

    SIMULATION = FALSE;
    IMPLEMENTATION = FALSE;
```

```
if (!hPrevInstance)
 {
 wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
 wndclass.lpfnWndProc   = WndProc ;
 wndclass.cbClsExtra    = 0 ;
 wndclass.cbWndExtra    = 0 ;
 wndclass.hInstance     = hInstance ;
 wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
 wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
 wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
 wndclass.lpszMenuName  = "MainMenu" ;
 wndclass.lpszClassName = szAppName ;
 RegisterClass (&wndclass) ;
 wndclass.lpfnWndProc   = WPStep;
 wndclass.lpszMenuName  = NULL ; //"StepMenu" ;
 wndclass.cbWndExtra                = 0;
 wndclass.hIcon                     = NULL;
 wndclass.lpszClassName  = szTimeResp;
 RegisterClass  (&wndclass);
 wndclass.lpfnWndProc   = FuzErrMemProc;
 wndclass.lpszMenuName  = NULL; //"ErrMenu" ;
 wndclass.lpszClassName  = szFuzErr;
 RegisterClass  (&wndclass);
 wndclass.lpfnWndProc   = FuzDerMemProc;
 wndclass.lpszMenuName  = NULL ; //"DerMenu" ;
 wndclass.lpszClassName  = szFuzDer;
 RegisterClass  (&wndclass);
 wndclass.lpszClassName  = szFuzOut;
 wndclass.lpszMenuName  = NULL; //"OutMenu" ;
 wndclass.lpfnWndProc   = FuzOutMemProc;
 RegisterClass  (&wndclass);
 wndclass.lpszClassName  = szFilter1;
 wndclass.lpszMenuName  = NULL; //"OutMenu" ;
 wndclass.lpfnWndProc   = FuzFilt1MemProc;
 RegisterClass  (&wndclass);
 wndclass.lpszClassName  = szFilter2;
 wndclass.lpszMenuName  = NULL; //"OutMenu" ;
 wndclass.lpfnWndProc   = FuzFilt2MemProc;
 RegisterClass  (&wndclass);
 }

hwnd = CreateWindow (szAppName,        // window class name
szAppName,          // window caption
WS_OVERLAPPEDWINDOW,    // window style
CW_USEDEFAULT,          // initial x position
CW_USEDEFAULT,          // initial y position
CW_USEDEFAULT,          // initial x size
CW_USEDEFAULT,          // initial y size
NULL,                   // parent window handle
NULL,                   // window menu handle
hInstance,              // program instance handle
NULL) ;                 // creation parameters
```

```
        ShowWindow (hwnd, SW_SHOWMAXIMIZED) ;
        UpdateWindow (hwnd) ;
        SendMessage(hwnd,WM_COMMAND,IDM_ERROR,0);
        SendMessage(hwnd,WM_COMMAND,IDM_DERIVATIVE,0);
        SendMessage(hwnd,WM_COMMAND,IDM_OUTRULES,0);
        SendMessage(hwnd,WM_COMMAND,IDM_STEP,0);
        SendMessage(hwnd,WM_COMMAND,IDM_FILTER,0);
        SendMessage(hwnd,WM_COMMAND,IDM_TILE,0);
        while (GetMessage (&msg, NULL, 0, 0))
        {
        TranslateMessage (&msg) ;
         DispatchMessage (&msg) ;
         }
        return msg.wParam ;
        }


void Gain_Button (HDC hdc,short ident)
        {
        POINT  pt[3];
        SelectObject(hdc,GetStockObject(BLACK_BRUSH));
        if (ident)
                {
                pt[0].x = BTN_WIDTH*.35; pt[0].y = BTN_HEIGHT*.35;
                pt[1].x = BTN_WIDTH*.65; pt[1].y = BTN_HEIGHT*.35;
                pt[2].x = BTN_WIDTH*.5; pt[2].y = BTN_HEIGHT*.65;
                }
        else
                {
                pt[0].x = BTN_WIDTH*.5; pt[0].y = BTN_HEIGHT*.35;
                pt[1].x = BTN_WIDTH*.35; pt[1].y = BTN_HEIGHT*.65;
                pt[2].x = BTN_WIDTH*.65; pt[2].y = BTN_HEIGHT*.65;
                }
        Polygon (hdc,pt,3);
        SelectObject(hdc,GetStockObject(BLACK_BRUSH));
        }

float xfabs(float x)
    {
    if (x>=0.0000) return x;
    return -1.00*x;
    }

long FAR PASCAL _export WndProc (HWND hwnd, UINT message, UINT wParam,
                                        LONG lParam)
        {
        short                   j;
        static dword            i;
        static HWND             hwndChild [] = { NULL,NULL,NULL,NULL,NULL,NULL};
```

```
RECT            rect;
HDC             hdc;
POINT           TLpt,BRpt;
int             Height,Width;
static float    x2,x1;
float           FuzzifiedERR,FuzzifiedROA,ERR,FUZMIN;
static int      Update_Interval;

switch (message)
        {
        case WM_COMMAND :
                hMainMenu = GetMenu (hwnd);
                GetClientRect(hwnd,&rect);
                switch (wParam)
                {
                case IDM_OPEN:
                        read_fuzzy_system(file_name , &fz);
                        &test_fuzzy_system);
                        InvalidateRect(hwndChild[1],&rect,TRUE);
                        SendMessage(hwndChild[1],WM_PAINT,0,0);
                        InvalidateRect(hwndChild[2],&rect,TRUE);
                        SendMessage(hwndChild[2],WM_PAINT,0,0);
                        return 0;
                case IDM_SAVE:
                        write_fuzzy_system(file_name , fz);
                        return 0;
                case IDM_SAVEAS:
                        return 0;
                case IDM_EXIT:
                        SendMessage(hwnd,WM_DESTROY,0,0);
                        return 0;
                case IDM_STEP:
                        hwndChild[0] = CreateWindow(szTimeResp, szTimeResp ,
                        WS_OVERLAPPEDWINDOW | WS_VISIBLE ,
                        rect.right/2,rect.bottom/2,rect.right/2,rect.bottom/2,
                        hwnd,NULL,
                        GetWindowWord(hwnd,GWW_HINSTANCE),NULL);
                        EnableMenuItem(hMainMenu,IDM_STEP,MF_GRAYED);
                        EnableMenuItem(hStepMenu,
                        IDM_START,MF_ENABLED);
                        TimeOpen = TRUE;
                        return 0;
                        case IDM_ERROR:
                        hwndChild[1] = CreateWindow(szFuzErr, szFuzErr ,
                        WS_OVERLAPPEDWINDOW | WS_VISIBLE ,
                        0,0,rect.right/2,rect.bottom/2,hwnd,NULL,
                        GetWindowWord(hwnd,
                        GWW_HINSTANCE),NULL);
                        EnableMenuItem(hMainMenu,
                        IDM_ERROR,MF_GRAYED);
                        ErrOpen = TRUE;
                        return 0;
```

```
case IDM_DERIVATIVE:
        hwndChild[2] = CreateWindow(szFuzDer, szFuzDer ,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE ,
        0,rect.bottom/2,rect.right/2,rect.bottom/2,hwnd,NULL,
        GetWindowWord(hwnd,GWW_HINSTANCE),NULL);
        EnableMenuItem(hMainMenu,
        IDM_DERIVATIVE,MF_GRAYED);
        DerOpen = TRUE;
        return 0;
case IDM_FILTER:
        hwndChild[4] = CreateWindow(szFilter1, szFilter1 ,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE ,
        0,rect.bottom/2,rect.right/2,rect.bottom/2,hwnd,NULL,
        GetWindowWord(hwnd,
        GWW_HINSTANCE),NULL);
        hwndChild[5] = CreateWindow(szFilter2, szFilter2 ,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE ,
        0,rect.bottom/2,rect.right/2,rect.bottom/2,hwnd,NULL,
        GetWindowWord(hwnd,
        GWW_HINSTANCE),NULL);
        EnableMenuItem(hMainMenu,
        IDM_FILTER,MF_GRAYED);
        FilterOpen = TRUE;
        return 0;
case IDM_OUTRULES:
        hwndChild[3] = CreateWindow(szFuzOut, szFuzOut ,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE ,
        rect.right/2,0,rect.right/2,rect.bottom/2,hwnd,NULL,
        GetWindowWord(hwnd,GWW_HINSTANCE),NULL);
        EnableMenuItem(hMainMenu,
        IDM_OUTRULES,MF_GRAYED);
        OutOpen = TRUE;
        return 0;
case IDM_STARTSIM:
        if (!em)
        {
        k = 0;
        em = new system_model;
        assert(em != NULL);
        }
        em->gai8.set_gain(ERR_GAIN);
        em->gai9.set_gain(ROA_GAIN);
        em->gai10.set_gain(FUZGAIN);
        em->gai5.set_gain(Err_Gain);
        em->gai6.set_gain(Der_Gain);
        em->gai7.set_gain(Out_Gain);
        em->ngai5.set_gain(Err_Gain);
        em->ngai6.set_gain(Der_Gain);
        em->ngai7.set_gain(Out_Gain);
        Update_Interval =
        ceil(System_Run_Time/System_Delta_Time/MAX_DATA);
        wrap_time=0;
```

```
            SetTimer(hwnd, 5, 50, NULL);
            EnableMenuItem(hMainMenu,
            IDM_STARTSIM,MF_GRAYED);
            EnableMenuItem(hMainMenu,
            IDM_RESETSIM,MF_ENABLED);
            EnableMenuItem(hMainMenu,
            IDM_PAUSESIM,MF_ENABLED);
            EnableMenuItem(hMainMenu,
            IDM_STARTIMP,MF_GRAYED);
            SIMULATION = TRUE;
            IMPLEMENTATION = FALSE;
            return 0;
case IDM_PAUSESIM:
            SIMULATION = FALSE;
            KillTimer(hwnd,5);
            EnableMenuItem(hMainMenu,
            IDM_RESETSIM,MF_ENABLED);
            EnableMenuItem(hMainMenu,
            IDM_PAUSESIM,MF_GRAYED);
            EnableMenuItem(hMainMenu,
            IDM_STARTSIM,MF_ENABLED);
            return 0;
case IDM_RESETSIM:
            k=0;
            InvalidateRect(hwndChild[0],NULL,TRUE);
            delete em;
            em = NULL;
            em = new system_model;
            assert(em != NULL);
            em->gai5.set_gain(Err_Gain);
            em->gai6.set_gain(Der_Gain);
            em->ngai5.set_gain(Err_Gain);
            em->gai8.set_gain(ERR_GAIN);
            em->gai9.set_gain(ROA_GAIN);
            em->gai10.set_gain(FUZGAIN);
            em->ngai6.set_gain(Der_Gain);
            em->gai7.set_gain(Out_Gain);
            em->ngai7.set_gain(Out_Gain);
            KillTimer(hwnd,5);
            EnableMenuItem(hMainMenu,
            IDM_RESETSIM,MF_GRAYED);
            EnableMenuItem(hMainMenu,
            IDM_PAUSESIM,MF_GRAYED);
            EnableMenuItem(hMainMenu,
            IDM_STARTSIM,MF_ENABLED);
            EnableMenuItem(hMainMenu,
            IDM_STARTIMP,MF_ENABLED);
            SIMULATION = FALSE;
            return 0;
case IDM_S_TIME:
            k = 0;
            et=0;
```

```
        Update_Interval = ceil(System_Run_Time/s_time/MAX_DATA);
        Cycles_per_Timer = 10000;
        init_engine_fuzzy_system(&fz);
        init_filt_fuzzy_system(&fzz);
        char buffer[100];
        ErrOpen=FALSE;
        DerOpen=FALSE;
        OutOpen=FALSE;
        FilterOpen=FALSE;
        TimeOpen=FALSE;
        IMPLEMENTATION = TRUE;
        clock_t start=clock();
        SendMessage(hwnd,WM_TIMER,0,0);
        s_time=(clock()-start)/CLK_TCK/Cycles_per_Timer;
        sprintf(buffer,"The sample time is %f\0",s_time);
        MessageBox(hwnd,buffer,"s_time",MB_OK);
        Cycles_per_Timer = 100;
        ErrOpen=TRUE;
        DerOpen=TRUE;
        OutOpen=TRUE;
        FilterOpen=TRUE;
        TimeOpen=TRUE;
        IMPLEMENTATION = FALSE;
        return 0;
case IDM_TILE :
        GetClientRect(hwnd,&rect);
        TLpt.x = 0;
        TLpt.y = 0;
        ClientToScreen(hwnd,&TLpt);
        BRpt.x = rect.right;
        BRpt.y = rect.bottom;
        ClientToScreen(hwnd,&BRpt);
        Height = rect.bottom/2;
        Width = rect.right/3;
        if (hwndChild[0]) MoveWindow (hwndChild[0],
        (BRpt.x-TLpt.x)/3*2+TLpt.x,(BRpt.y-TLpt.y)/2+TLpt.y,Width,Height,TRUE);
        if (hwndChild[1]) MoveWindow (hwndChild[1],
        TLpt.x,TLpt.y,Width,Height,TRUE);
        if (hwndChild[2]) MoveWindow (hwndChild[2],
        TLpt.x,(BRpt.y-TLpt.y)/2+TLpt.y,Width,Height,TRUE);
        if (hwndChild[3]) MoveWindow (hwndChild[3],
        (BRpt.x-TLpt.x)/3*2+TLpt.x,TLpt.y,Width,Height,TRUE);
        if (hwndChild[4]) MoveWindow (hwndChild[4],
        (BRpt.x-TLpt.x)/3+TLpt.x,(BRpt.y-TLpt.y)/2+TLpt.y,Width,Height,TRUE);
        if (hwndChild[5]) MoveWindow (hwndChild[5],
        (BRpt.x-TLpt.x)/3+TLpt.x,TLpt.y,Width,Height,TRUE);
        return 0;
case IDM_HELP:
        return 0;
case IDM_ABOUT:
        return 0;
default:
```

```
        return 0;
}
case WM_CREATE :
        return 0;
case WM_TIMER :
        //MessageBeep (0);
        hdc = GetDC(hwnd);
        GetClientRect(hwnd,&rect);
        if (hdc)
        {
        // get next page of data
        for (j=0;j<Cycles_per_Timer;j++)
        {
        // update model
        if (SIMULATION)
        {
        if (!em->sim.update())
        {
        KillTimer(hwnd,1);
        SendMessage(hwnd,WM_COMMAND,IDM_RESETSIM,0);
        return 0;
        }
        // Get The New Counter And Desired Block Outputs...
        i  = em->sim.iterations() ;
        et = em->sim.elapsed_time();
        SendMessage(hwnd,WM_COMMAND,IDM_RESETSIM,0);
        }
        // Update The Screen Every display_step Samples...
        if ( (i % Update_Interval ) == 0 )
        {
        // get output data from simulator
        if (SIMULATION)
        {
        data1[k] = em->gai7.output(et) - em->gai10.output(et);
        data2[k] = em->ngai7.output(et);
        data3[k] = em->pid.output(et);
        data4[k] = em->sum1.output(et);
        }
        }
        ReleaseDC (hwnd,hdc);
        }
        return 0;
case WM_DESTROY:
        DeleteObject(hPen1);
        DeleteObject(hPen2);
        DeleteObject(hPen3);
        DeleteObject(hPen4);
        DeleteObject(hPen5);
        delete  data_et;
        delete  data1;
        delete  data2;
        delete  data3;
```

```
        delete  data4;
        PostQuitMessage (0) ;
        return 0 ;
        }
        return DefWindowProc (hwnd, message, wParam, lParam) ;
        }

long FAR PASCAL _export WPStep (HWND hwnd, UINT message, UINT wParam,
                                LONG lParam)
{
const float    border=.1;
short                                   j;
static int                      x_new,x_old,ref1,out1,out2,out3;
TEXTMETRIC                      tm;
HDC                hdc ;
static HRGN    hrgn;
PAINTSTRUCT                     ps ;
static int                 i;
static float               w_height;
static RECT        rect,rc;
static float   max_err=0, max_err2=0;;
switch (message)
{
case WM_COMMAND :
        hStepMenu = GetMenu (hwnd);
        GetClientRect(hwnd,&rect);
        switch (wParam)
        {
case IDM_CLOSE:
        SendMessage(hwnd,WM_CLOSE,0,0);
        return 0;
case IDM_START:
        TimeOpen = TRUE;
        EnableMenuItem(hStepMenu,IDM_START,MF_GRAYED);
        EnableMenuItem(hStepMenu,IDM_STOP,MF_ENABLED);
        return 0;
case IDM_STOP:
        TimeOpen = FALSE;
        EnableMenuItem(hStepMenu,IDM_STOP,MF_GRAYED);
        EnableMenuItem(hStepMenu,IDM_START,MF_ENABLED);
        return 0;
default:
        return 0;
        }
case WM_CREATE :
        hdc = GetDC(hwnd);
        SelectObject (hdc,GetStockObject(BLACK_PEN));
        GetTextMetrics ( hdc, &tm);
        ReleaseDC (hwnd,hdc);
        return 0;
case WM_MOVE:
        // same code as paint message
```

```
case WM_PAINT:
        hdc = BeginPaint (hwnd, &ps) ;
        GetClientRect(hwnd,&rect);
        hrgn = CreateRectRgn(rect.right*border+4,rect.bottom*border,rect.right*(1-
        border),rect.bottom*(1-border)-4);
        GetRgnBox(hrgn,&rect);
        //SelectClipRgn(hdc,hrgn);
        x_new = rect.left;
        w_height =  rect.bottom-rect.top;
        ref1 = rect.top+(max_x-data1[0])/(max_x-min_x)*w_height;
        out1 = ref1;
        out2 = ref1;
        out3 = ref1;
        // draw and label axis
        SelectObject (hdc,hPen4);
        MoveTo (hdc,rect.left,rect.top);
        LineTo (hdc,rect.left,rect.bottom);
        LineTo (hdc,rect.right,rect.bottom);
        LineTo (hdc,rect.right,rect.top);
        LineTo (hdc,rect.left,rect.top);
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        //GetClientRect(hwnd,&rect);
        HFONT hfont, hfontOld;
        LOGFONT lf;
        lf.lfHeight = rect.right*.04;
        lf.lfWidth = 0;
    lf.lfEscapement = 900;
    lf.lfOrientation = 0; lf.lfWeight = FW_NORMAL; lf.lfItalic = 0; lf.lfUnderline =0; lf.lfStrikeOut = 0;
    lf.lfCharSet = ANSI_CHARSET; lf.lfOutPrecision = OUT_DEFAULT_PRECIS; lf.lfClipPrecision =
    CLIP_DEFAULT_PRECIS; lf.lfQuality = DEFAULT_QUALITY; lf.lfPitchAndFamily =
    FF_DONTCARE|DEFAULT_PITCH; strcpy(lf.lfFaceName,"Arial"); hfont = CreateFontIndirect(&lf);
    hfontOld = SelectObject(hdc, hfont); char yaxis[] = "Speed (rpm)";

    SetTextAlign(hdc,TA_CENTER|TA_BASELINE);

    TextOut(hdc, rect.left/2,(rect.bottom+rect.top)/2, yaxis, strlen(yaxis)); char xaxis[] = "Time (s)"  ;

    DeleteObject(hfont); lf.lfEscapement = 0; hfont = CreateFontIndirect(&lf);

    SelectObject(hdc, hfont);

    SetTextAlign(hdc,TA_CENTER|TA_LEFT);

    TextOut(hdc, (rect.left+rect.right)/2, rect.bottom*(1+border/2), xaxis, strlen(xaxis));

    DeleteObject(hfont);

    SetTextAlign(hdc,TA_BOTTOM|TA_LEFT);

    lf.lfHeight = rect.right*.02;
    lf.lfWidth = 0;
    hfont = CreateFontIndirect(&lf);
    SelectObject(hdc, hfont);
    //GetRgnBox(hrgn,&rect);
    char xmark[50];
    char ymark[50];
```

```
// create ticks and label each axis
for (j=0;j<=6;j++)
    {
     SelectObject (hdc,GetStockObject (BLACK_PEN));
     MoveTo (hdc,rect.left,rect.top+w_height/6*float(j));
     if (j!=6) LineTo (hdc,rect.left+(rect.right-rect.left)*.02,rect.top+w_height/6*float(j));
    //grids for y-axis
    SelectObject (hdc,hPen5);
    MoveTo (hdc,rect.left,rect.top+w_height/6*float(j));
     if (j!=6 && j!=0) LineTo (hdc,rect.right,rect.top+w_height/6*float(j));
    //ticks for x-axis
    SelectObject (hdc,GetStockObject (BLACK_PEN));
    MoveTo (hdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom);
    if (j!=0) LineTo (hdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom-w_height*.02);
    //grids for x-axis
    SelectObject (hdc,hPen5);
    MoveTo (hdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom);
     if (j!=0 && j!=6) LineTo (hdc,rect.left+(rect.right-rect.left)/6*float(j),rect.top);
    //labels for x-axis
    SetTextAlign(hdc,TA_CENTER|TA_BOTTOM);
    TextOut(hdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom*(1+border/2),
    xmark,sprintf(xmark,"%4.1f",System_Run_Time/6*float(j)));
    //labels for y-axis
    SetTextAlign(hdc,TA_CENTER|TA_BASELINE);
    TextOut(hdc,rect.left/4*3,rect.top+w_height/6*float(6-j),
     ymark,sprintf(ymark,"%3.0f", min_x + (max_x-min_x)/6*float(j) ));
    }

SelectObject(hdc, hfontOld);
DeleteObject(hfont);
SelectObject (hdc,GetStockObject (BLACK_PEN));

for (j=0;j<k;j++)
     {
     //draw referance
     MoveTo (hdc,x_new,ref1);
     x_old = x_new;
     x_new = rect.left+int(data_et[j]/System_Run_Time*(rect.right-rect.left));
     ref1 = rect.top+(max_x-data1[j])/(max_x-min_x)*w_height;
     LineTo (hdc,x_new,ref1);
     //draw system 1 or honda
     //SelectObject (hdc,hPen1);
     MoveTo (hdc,x_old,out1);
     out1 = rect.top+(max_x-data2[j])/(max_x-min_x)*w_height;
     LineTo (hdc,x_new,out1);
     // draw pid output
     //SelectObject (hdc,hPen2);
     MoveTo (hdc,x_old,out2);
     out2 = rect.top+(max_x-data4[j])/(max_x-min_x)*w_height;
     LineTo (hdc,x_new,out2);
```

```
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        }
         ReleaseDC(hwnd,hdc);
        EndPaint (hwnd, &ps) ;
        return 0 ;
case WM_KEYUP:
        if (wParam == VK_F1)
        {
        HANDLE hmf;
        HDC mdc;
        GLOBALHANDLE hgmem;
        LPMETAFILEPICT mfp;
        mdc = CreateMetaFile (NULL) ;
        if (mdc == NULL)
                {
                MessageBox(hwnd,"Meta Create Error","Error",MB_OK);
                return 0;
                }
        GetClientRect(hwnd,&rect);
        hrgn = CreateRectRgn(rect.right*border+4,rect.bottom*border,rect.right*(1-
                border),rect.bottom*(1-border)-4);
         GetRgnBox(hrgn,&rect);
         //SelectClipRgn(mdc,hrgn);
         x_new = rect.left;
         w_height =  rect.bottom-rect.top;
         ref1 = rect.top+(max_x-data1[0])/(max_x-min_x)*w_height;
         out1 = ref1;
         out2 = ref1;
         out3 = ref1;
         // draw and label axis
          SelectObject (mdc,hPen4);
          MoveTo (mdc,rect.left,rect.top);
          LineTo (mdc,rect.left,rect.bottom);
          LineTo (mdc,rect.right,rect.bottom);
          LineTo (mdc,rect.right,rect.top);
          LineTo (mdc,rect.left,rect.top);
          SelectObject (mdc,GetStockObject (BLACK_PEN));
          //GetClientRect(hwnd,&rect);
           HFONT hfont, hfontOld;
           LOGFONT lf;
            lf.lfHeight = rect.right*.04;
            lf.lfWidth = 0;
            lf.lfEscapement = 900;
            lf.lfOrientation = 0;
            lf.lfWeight = FW_NORMAL;
            lf.lfItalic = 0;
            lf.lfUnderline =0;
            lf.lfStrikeOut = 0;
            lf.lfCharSet = ANSI_CHARSET;
            lf.lfOutPrecision = OUT_DEFAULT_PRECIS;
            lf.lfClipPrecision = CLIP_DEFAULT_PRECIS;
            lf.lfQuality = DEFAULT_QUALITY;
```

```
lf.lfPitchAndFamily = FF_DONTCARE|DEFAULT_PITCH;
strcpy(lf.lfFaceName,"Arial");
hfont = CreateFontIndirect(&lf);
hfontOld = SelectObject(mdc, hfont);
char yaxis[] = "Speed (rpm)";
SetTextAlign(mdc,TA_CENTER|TA_BASELINE);
TextOut(mdc, rect.left/2,(rect.bottom+rect.top)/2, yaxis, strlen(yaxis));
char xaxis[] = "Time (s)" ;
DeleteObject(hfont);
lf.lfEscapement = 0;
hfont = CreateFontIndirect(&lf);
SelectObject(mdc, hfont);
SetTextAlign(mdc,TA_CENTER|TA_LEFT);
TextOut(mdc, (rect.left+rect.right)/2, rect.bottom*(1+border/2), xaxis, strlen(xaxis));
DeleteObject(hfont);
SetTextAlign(mdc,TA_BOTTOM|TA_LEFT);
lf.lfHeight = rect.right*.02;
lf.lfWidth = 0;
hfont = CreateFontIndirect(&lf);
SelectObject(mdc, hfont);
//GetRgnBox(hrgn,&rect);
char xmark[50];
char ymark[50];
// create ticks and label each axis
for (j=0;j<=6;j++)
{
 //ticks for y-axis
 SelectObject (mdc,GetStockObject (BLACK_PEN));
 MoveTo (mdc,rect.left,rect.top+w_height/6*float(j));
 if (j!=6) LineTo (mdc,rect.left+(rect.right-rect.left)*.02,rect.top+w_height/6*float(j));
  //grids for y-axis
 SelectObject (mdc,hPen5);
 MoveTo (mdc,rect.left,rect.top+w_height/6*float(j));
 if (j!=6 && j!=0) LineTo (mdc,rect.right,rect.top+w_height/6*float(j));
  //ticks for x-axis
  SelectObject (mdc,GetStockObject (BLACK_PEN));
  MoveTo (mdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom);
  if (j!=0) LineTo (mdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom-w_height*.02);
  //grids for x-axis
  SelectObject (mdc,hPen5);
  MoveTo (mdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom);
  if (j!=0 && j!=6) LineTo (mdc,rect.left+(rect.right-rect.left)/6*float(j),rect.top);
  //labels for x-axis
  SetTextAlign(mdc,TA_CENTER|TA_BOTTOM);
  TextOut(mdc,rect.left+(rect.right-rect.left)/6*float(j),rect.bottom*(1+border/2),
   xmark,sprintf(xmark,"%4.1f",System_Run_Time/6*float(j)));
   //labels for y-axis
   SetTextAlign(mdc,TA_CENTER|TA_BASELINE);
   TextOut(mdc,rect.left/4*3,rect.top+w_height/6*float(6-j),
   ymark,sprintf(ymark,"%3.0f", min_x + (max_x-min_x)/6*float(j) ));
    }
```

```
                SelectObject(mdc, hfontOld);
                DeleteObject(hfont);
                SelectObject (mdc,GetStockObject (BLACK_PEN));
                for (j=0;j<k;j++)
                  {
                  //draw referance
                  MoveTo (mdc,x_new,ref1);
                  x_old = x_new;
                  x_new = rect.left+int(data_et[j]/System_Run_Time*(rect.right-rect.left));
                  ref1 = rect.top+(max_x-data1[j])/(max_x-min_x)*w_height;
                  LineTo (mdc,x_new,ref1);
                  //draw system 1 or honda
                  SelectObject (mdc,hPen1);
                  MoveTo (mdc,x_old,out1);
                  out1 = rect.top+(max_x-data2[j])/(max_x-min_x)*w_height;
                  LineTo (mdc,x_new,out1);
                  //draw system 2
                  SelectObject (mdc,hPen3);
                  MoveTo (mdc,x_old,out3);
                  out3 = rect.top+(max_x-data3[j])/(max_x-min_x)*w_height;
                  LineTo (mdc,x_new,out3);
                  // draw pid output
                  SelectObject (mdc,hPen2);
                  MoveTo (mdc,x_old,out2);
                  out2 = rect.top+(max_x-data4[j])/(max_x-min_x)*w_height;
                  LineTo (mdc,x_new,out2);
                  SelectObject (mdc,GetStockObject (BLACK_PEN));
                  }
                hmf = CloseMetaFile(mdc);
                hgmem = GlobalAlloc(GHND, (DWORD) sizeof(METAFILEPICT));
                mfp = (LPMETAFILEPICT) GlobalLock(hgmem);
                mfp->mm = MM_TEXT;
                mfp->xExt=1500;
                mfp->yExt=1200;
                mfp->hMF = hmf;
                GlobalUnlock(hgmem);
                OpenClipboard(hwnd);
                EmptyClipboard();
                SetClipboardData(CF_METAFILEPICT,hgmem);
                CloseClipboard();
                }
              return 0 ;
case WM_TIMER :
        //MessageBeep (0);
        hdc = GetDC(hwnd);
        if (hdc)
        {
        for (j=i;j<k;j++)
        {
        MoveTo (hdc,x_new,ref1);
        x_old = x_new;
        x_new = rect.left+data_et[j]/System_Run_Time*(rect.right-rect.left);
```

```
        ref1 = rect.top+(max_x-data1[j])/(max_x-min_x)*w_height;
        LineTo (hdc,x_new,ref1);
        SelectObject (hdc,hPen1);
        MoveTo (hdc,x_old,out1);
        out1 = rect.top+(max_x-data2[j])/(max_x-min_x)*w_height;
        LineTo (hdc,x_new,out1);
        MoveTo (hdc,x_old,out3);
        out3 = rect.top+(max_x-data3[j])/(max_x-min_x)*w_height;
        LineTo (hdc,x_new,out3);
        // draw pid output
//      SelectObject (hdc,hPen2);
        MoveTo (hdc,x_old,out2);
        out2 = rect.top+(max_x-data4[j])/(max_x-min_x)*w_height;
        LineTo (hdc,x_new,out2);

        //print current throttle position
        char junk[50];
        TextOut(hdc,rect.left+30,rect.top+30,junk,sprintf(junk,"Time Delay  = %.4f s",eng_delay));
        TextOut(hdc,rect.left+30,rect.top+50,junk,sprintf(junk,"Load = %.1f ft-lb",ld1_mag));
        TextOut(hdc,rect.left+30,rect.top+70,junk,sprintf(junk,"P.V. Fact. = %.1f",par_var));
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        }
        ReleaseDC (hwnd,hdc);
        i=k;
        }
        return 0;

case WM_CLOSE:
        TimeOpen = FALSE;
        DestroyWindow (hwnd) ;
        EnableMenuItem(hMainMenu,IDM_STEP,MF_ENABLED);
        EnableMenuItem(hStepMenu,IDM_STOP,MF_GRAYED);
        return 0 ;
        }
 return DefWindowProc (hwnd, message, wParam, lParam) ;
}

long FAR PASCAL _export FuzErrMemProc (HWND hwnd, UINT message, UINT wParam,
                        LONG lParam)
        {
        #define minx -2.0
        #define maxx 2.0
        #define miny 0.0
        #define maxy 1.0

        int                     i,j,k=0;
        HDC                     hdc ;
        PAINTSTRUCT             ps ;
        static short            cxChar,cyChar;
        TEXTMETRIC              tm;
        char                    char_str[10];
        static RECT             rect;
```

```
HRGN            hrgn;
DWORD           dword;
float           Inp_Err;
static POINT    AR1,AR2,AR3,AR4;
static HWND     hwndSmaller,hwndLarger;
LPDRAWITEMSTRUCT        lpdis;
char            szbuffer[50];
static float    MEM_FCN[3][3];


switch (message)
        {
        case WM_COMMAND :
                hErrMenu = GetMenu (hwnd);
                GetClientRect(hwnd,&rect);
                switch (wParam)
                {
                case IDC_SMALLER :
                        hdc = GetDC(hwnd);
                        Err_Gain -= .001;
                        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.3f ",Err_Gain));
                        em->gai5.set_gain(Err_Gain);
                        em->ngai5.set_gain(Err_Gain);
                        ReleaseDC(hwnd,hdc);
                        return 0;
                case IDC_LARGER :
                        hdc = GetDC(hwnd);
                        Err_Gain += .001;
                        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.3f ",Err_Gain));
                        em->gai5.set_gain(Err_Gain);
                        em->ngai5.set_gain(Err_Gain);
                        ReleaseDC(hwnd,hdc);
                        return 0;
                case IDM_START:
                        ErrOpen = TRUE;
                        EnableMenuItem(hErrMenu,IDM_START,MF_GRAYED);
                        EnableMenuItem(hErrMenu,IDM_STOP,MF_ENABLED);
                        return 0;
                case IDM_STOP:
                        ErrOpen = FALSE;
                        EnableMenuItem(hErrMenu,IDM_STOP,MF_GRAYED);
                        EnableMenuItem(hErrMenu,IDM_START,MF_ENABLED);
                        return 0;
                case IDM_CLOSE:
                        SendMessage(hwnd,WM_CLOSE,0,0);
                        return 0;
                case IDM_GAIN:
                        return 0;
                default:
                        return 0;
                }
        case WM_DRAWITEM :
```

```
                lpdis= (LPDRAWITEMSTRUCT) lParam;
                FillRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(LTGRAY_BRUSH));
                FrameRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(BLACK_BRUSH));
                switch (lpdis->CtlID)
                        {
                        case IDC_SMALLER:
                                Gain_Button(lpdis->hDC,UP);
                                break;
                        case IDC_LARGER :
                                Gain_Button(lpdis->hDC,DOWN),
                                break;
                        }
                return 0;
        case WM_SIZE :
                MoveWindow                                      (hwndSmaller,10,
                20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
                MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
                return 0;
        case WM_CREATE :
                hdc= GetDC(hwnd);
                hwndSmaller = CreateWindow("button","",
                 WS_CHILD | WS_VISIBLE | BS_OWNERDRAW, 0,0,
                 BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_SMALLER,hInst,NULL);
                hwndLarger = CreateWindow("button","",
                WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,0,0,
                BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_LARGER,hInst,NULL);
                Err_Gain = em->gai5.get_gain();
                GetTextMetrics ( hdc, &tm);
                cxChar = tm.tmAveCharWidth;
                cyChar = tm.tmHeight + tm.tmExternalLeading;
                ReleaseDC(hwnd,hdc);
                return 0;
        case WM_PAINT :
                hdc = BeginPaint (hwnd, &ps) ;
                GetClientRect(hwnd, &rect);
                MoveWindow
(hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
                MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
                TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.3f ",Err_Gain));
                SelectObject (hdc,hPen1);
                for (j=0;j<fz.no_of_inp_regions;j++)
                {
                if ((j == 0) )
                {
                MoveTo( hdc,.1*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
                LineTo(hdc,(fz.inp_mem_fns[0][j].a-minx)/(maxx-
                minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
                LineTo(hdc,(fz.inp_mem_fns[0][j].b-minx)/(maxx-
                minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
                }
                else if (j == fz.no_of_inp_regions-1)
                {
```

```
        MoveTo( hdc,(fz.inp_mem_fns[0][j].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[0][j].b-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,.9*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        }
        else
        {
        MEM_FCN[j-1][0] = fz.inp_mem_fns[0][j].a;
        MEM_FCN[j-1][1] = fz.inp_mem_fns[0][j].b;
        MEM_FCN[j-1][2] = fz.inp_mem_fns[0][j].d;
        MoveTo( hdc,(fz.inp_mem_fns[0][j].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[0][j].b-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[0][j].d-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        }
        } /*  end j  */
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        MoveTo( hdc,.05*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,.95*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        MoveTo( hdc,.5*rect.right,(rect.bottom*.05-miny)/(maxy-miny));
        LineTo(hdc,.5*rect.right,(rect.bottom*.95-miny)/(maxy-miny));
        for (j=0;j<=4;j++)
        {
        MoveTo( hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.88-
        miny)/(maxy-miny));
        LineTo(hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
        miny)/(maxy-miny));
        MoveTo( hdc,(.25*j-minx)/(maxx-inx)*rect.right,(rect.bottom*.88-ny)/(maxy-
        miny));
        LineTo(hdc,(.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
        miny)/(maxy-miny));
        }
        EndPaint (hwnd, &ps);
        return 0;
case WM_TIMER :
        hdc = GetDC(hwnd);
        SelectObject (hdc,GetStockObject (WHITE_PEN));
        /*r (j=0;j<3;j++)
        {
        MoveTo( hdc,(MEM_FCN[j][0]-minx)/(maxx-minx)*rect.right,(rect.bottom*.9-
        miny)/(maxy-miny));
        LineTo(hdc,(MEM_FCN[j][1]-minx)/(maxx-minx)*rect.right,(rect.bottom*.1-
        miny)/(maxy-miny));
        LineTo(hdc,(MEM_FCN[j][2]-minx)/(maxx-minx)*rect.right,(rect.bottom*.9-
        miny)/(maxy-miny));
        }
        for (j=1;j<4;j++)
        {
        MEM_FCN[j-1][0] = fz.inp_mem_fns[0][j].a;
```

```
                        MEM_FCN[j-1][1] = fz.inp_mem_fns[0][j].b;
                        MEM_FCN[i-1][2] = fz.inp_mem_fns[0][j].d;
                        } */
                        MoveTo( hdc,AR1.x,AR1.y);
                        LineTo(hdc,AR2.x,AR2.y);
                        MoveTo( hdc,AR3.x,AR3.y);
                        LineTo(hdc,AR2.x,AR2.y);
                        LineTo(hdc,AR4.x,AR4.y);
                        if (SIMULATION) Inp_Err = em->gai5.output(et);
                        if (IMPLEMENTATION) Inp_Err = input[in_x];
                        SelectObject (hdc,hPen1);
                        /*for (j=0;j<3;j++)
                        {
                        MoveTo( hdc,(MEM_FCN[j][0]-minx)/(maxx-minx)*rect.right,(rect.bottom*.9-
                        miny)/(maxy-miny));
                        LineTo(hdc,(MEM_FCN[j][1]-minx)/(maxx-minx)*rect.right,(rect.bottom*.1-
                        miny)/(maxy-miny));
                        LineTo(hdc,(MEM_FCN[j][2]-minx)/(maxx-minx)*rect.right,(rect.bottom*.9-
                        miny)/(maxy-miny));
                        } */
                        SelectObject (hdc,GetStockObject (BLACK_PEN));
                        AR1.x = (Inp_Err-minx)/(maxx-minx)*rect.right*.97;
                        AR1.y = (rect.bottom*.94-miny)/(maxy-miny);
                        AR2.x = (Inp_Err-minx)/(maxx-minx)*rect.right;
                        AR2.y = (rect.bottom*.9-miny)/(maxy-miny)+1;
                        AR3.x = (Inp_Err-minx)/(maxx-minx)*rect.right*1.03;
                        AR3.y = (rect.bottom*.94-miny)/(maxy-miny);
                        AR4.x = (Inp_Err-minx)/(maxx-minx)*rect.right;
                        AR4.y = (rect.bottom*.98-miny)/(maxy-miny);
                        MoveTo( hdc,AR1.x,AR1.y);
                        LineTo(hdc,AR2.x,AR2.y);
                        MoveTo( hdc,AR3.x,AR3.y);
                        LineTo(hdc,AR2.x,AR2.y);
                        LineTo(hdc,AR4.x,AR4.y);
                        ReleaseDC (hwnd,hdc);
                        return 0;
                case WM_CLOSE:
                        ErrOpen = FALSE;
                        DestroyWindow (hwnd) ;
                        EnableMenuItem(hMainMenu,IDM_ERROR,MF_ENABLED);
                        return 0 ;
                }
        return DefWindowProc (hwnd, message, wParam, lParam) ;
        }

long FAR PASCAL _export FuzDerMemProc (HWND hwnd, UINT message, UINT wParam,
                                LONG lParam)
        {
        int                     i,j,k=0;
        HDC                     hdc ;
        PAINTSTRUCT             ps ;
        static short            cxChar,cyChar;
```

```
TEXTMETRIC          tm;
char                char_str[10];
static RECT         rect;
HRGN                hrgn;
DWORD               dword;
float               Inp_Err;
static POINT        AR1,AR2,AR3,AR4;
static HWND         hwndSmaller,hwndLarger;
LPDRAWITEMSTRUCT        lpdis;
char                szbuffer[50];


switch (message)
        {
        case WM_COMMAND :
                hDerMenu = GetMenu (hwnd);
                GetClientRect(hwnd,&rect);
                switch (wParam)
                {
                case IDC_SMALLER :
                        hdc = GetDC(hwnd);
                        Der_Gain -= .0001;
                        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",Der_Gain));
                        em->gai6.set_gain(Der_Gain);
                        em->ngai6.set_gain(Der_Gain);
                        ReleaseDC(hwnd,hdc);
                        return 0;
                case IDC_LARGER :
                        hdc = GetDC(hwnd);
                        Der_Gain += .0001;
                        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",Der_Gain));
                        em->gai6.set_gain(Der_Gain);
                        em->ngai6.set_gain(Der_Gain);
                        ReleaseDC(hwnd,hdc);
                        return 0;
                case IDM_START:
                        DerOpen = TRUE;
                        EnableMenuItem(hDerMenu,IDM_START,MF_GRAYED);
                        EnableMenuItem(hDerMenu,IDM_STOP,MF_ENABLED);
                        return 0;
                case IDM_STOP:
                        DerOpen = FALSE;
                        EnableMenuItem(hDerMenu,IDM_STOP,MF_GRAYED);
                        EnableMenuItem(hDerMenu,IDM_START,MF_ENABLED);
                        return 0;
                case IDM_CLOSE:
                        SendMessage(hwnd,WM_CLOSE,0,0);
                        return 0;
                case IDM_GAIN:
                        return 0;
                default:
                        return 0;
```

```
        }
case WM_DRAWITEM :
        lpdis= (LPDRAWITEMSTRUCT) lParam;
        FillRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(LTGRAY_BRUSH));
        FrameRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(BLACK_BRUSH));
        switch (lpdis->CtlID)
        {
        case IDC_SMALLER:
                Gain_Button(lpdis->hDC,UP);
                break;
        case IDC_LARGER :
                Gain_Button(lpdis->hDC,DOWN);
                break;
        }
        return 0;
case WM_SIZE :
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        return 0;
case WM_CREATE :
        hdc= GetDC(hwnd);
        hwndSmaller = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW, 0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_SMALLER,hInst,NULL);
        hwndLarger = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_LARGER,hInst,NULL);
        Der_Gain = em->gai6.get_gain();
        GetTextMetrics ( hdc, &tm);
        cxChar = tm.tmAveCharWidth;
        cyChar = tm.tmHeight + tm.tmExternalLeading;
        ReleaseDC(hwnd,hdc);
        return 0;
case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;
        GetClientRect(hwnd, &rect);
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",Der_Gain));
        SelectObject (hdc,hPen1);
        for (j=0;j<fz.no_of_inp_regions;j++)
        {
        if ((j == 0) )
        {
        MoveTo( hdc,.1*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[1][j].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[1][j].b-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        }
```

```
        else if (j == fz.no_of_inp_regions-1)
        {
        MoveTo( hdc,(fz.inp_mem_fns[1][j].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[1][j].b-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,.9*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        }
        else
        {
        MoveTo( hdc,(fz.inp_mem_fns[1][j].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[1][j].b-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[1][j].c-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,(fz.inp_mem_fns[1][j].d-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        }
        } /* end j */
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        MoveTo( hdc,.05*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,.95*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        MoveTo( hdc,.5*rect.right,(rect.bottom*.05-miny)/(maxy-miny));
        LineTo(hdc,.5*rect.right,(rect.bottom*.95-miny)/(maxy-miny));
        for (j=0;j<=4;j++)
        {
        MoveTo( hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.88-
        miny)/(maxy-miny));
        LineTo(hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
        miny)/(maxy-miny));
        MoveTo( hdc,(.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.88-
        miny)/(maxy-miny));
        LineTo(hdc,(.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
        miny)/(maxy-miny));
        }
        EndPaint (hwnd, &ps);
        return 0;
case WM_TIMER :
        hdc = GetDC(hwnd);
        SelectObject (hdc,GetStockObject (WHITE_PEN));
        MoveTo( hdc,AR1.x,AR1.y);
        LineTo(hdc,AR2.x,AR2.y);
        MoveTo( hdc,AR3.x,AR3.y);
        LineTo(hdc,AR2.x,AR2.y);
        LineTo(hdc,AR4.x,AR4.y);
        if (SIMULATION) Inp_Err = em->gai6.output(et);
        if (IMPLEMENTATION) Inp_Err = input[in_x_dot];
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        AR1.x = (Inp_Err-minx)/(maxx-minx)*rect.right*.97;
        AR1.y = (rect.bottom*.94-miny)/(maxy-miny);
        AR2.x = (Inp_Err-minx)/(maxx-minx)*rect.right;
```

```
                    AR2.y = (rect.bottom*.9-miny)/(maxy-miny)+1;
                    AR3.x = (Inp_Err-minx)/(maxx-minx)*rect.right*1.03;
                    AR3.y = (rect.bottom*.94-miny)/(maxy-miny);
                    AR4.x = (Inp_Err-minx)/(maxx-minx)*rect.right;
                    AR4.y = (rect.bottom*.98-miny)/(maxy-miny);
                    MoveTo( hdc,AR1.x,AR1.y);
                    LineTo(hdc,AR2.x,AR2.y);
                    MoveTo( hdc,AR3.x,AR3.y);
                    LineTo(hdc,AR2.x,AR2.y);
                    LineTo(hdc,AR4.x,AR4.y);
                    ReleaseDC (hwnd,hdc);
                    return 0;

            case WM_CLOSE:
                    DerOpen = FALSE;
                    DestroyWindow (hwnd) ;
                    EnableMenuItem(hMainMenu,IDM_DERIVATIVE,MF_ENABLED);
                    return 0 ;
            }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
    }


long FAR PASCAL _export FuzOutMemProc (HWND hwnd, UINT message, UINT wParam,
                    LONG lParam)
    {
    #define omaxx 1.5
    #define ominx -1.5
    #define ominy 0.0
    #define omaxy 1.0


    HDC                 hdc ;
    PAINTSTRUCT         ps ;
    static POINT        pt[25][2],AR1,AR2,AR3,AR4;
    int                 i;
    float               fuzzy_cs;
    static RECT         rect;
    char                szbuffer[50];
    static HWND         hwndSmaller,hwndLarger;
    LPDRAWITEMSTRUCT        lpdis;


    switch (message)
            {
            case WM_COMMAND :
                    hOutMenu = GetMenu (hwnd);
                    GetClientRect(hwnd,&rect);
                    switch (wParam)
                    {
                    case IDC_SMALLER :
                            hdc = GetDC(hwnd);
```

```
                Out_Gain -= 10;
                TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.3f ",Out_Gain));
                em->gai7.set_gain(Out_Gain);
                em->ngai7.set_gain(Out_Gain);
                ReleaseDC(hwnd,hdc);
                return 0;
        case IDC_LARGER :
                hdc = GetDC(hwnd);
                Out_Gain += 10;
                TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.3f ",Out_Gain));
                em->gai7.set_gain(Out_Gain);
                em->ngai7.set_gain(Out_Gain);
                ReleaseDC(hwnd,hdc);
                return 0;
        case IDM_START:
                OutOpen = TRUE;
                EnableMenuItem(hOutMenu,IDM_START,MF_GRAYED);
                EnableMenuItem(hOutMenu,IDM_STOP,MF_ENABLED);
                return 0;
        case IDM_STOP:
                OutOpen = FALSE;
                EnableMenuItem(hOutMenu,IDM_STOP,MF_GRAYED);
                EnableMenuItem(hOutMenu,IDM_START,MF_ENABLED);
                return 0;
        case IDM_CLOSE:
                SendMessage(hwnd,WM_CLOSE,0,0);
                return 0;
        case IDM_GAIN:
                return 0;
        default:
                return 0;
        }
case WM_DRAWITEM :
        lpdis= (LPDRAWITEMSTRUCT) lParam;
        FillRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(LTGRAY_BRUSH));
        FrameRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(BLACK_BRUSH));
        switch (lpdis->CtlID)
        {
        case IDC_SMALLER:
                Gain_Button(lpdis->hDC,UP);
                break;
        case IDC_LARGER :
                Gain_Button(lpdis->hDC,DOWN);
                break;
        }
        return 0;
case WM_SIZE :
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        return 0;
case WM_CREATE :
```

```
        hwndSmaller = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW, 0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_SMALLER,hInst,NULL);
        hwndLarger = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_LARGER,hInst,NULL);
        Out_Gain = em->gai7.get_gain();
        return 0;
case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;
        GetClientRect(hwnd,&rect);
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.3f ",Out_Gain));
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        MoveTo( hdc,.05*rect.right,(rect.bottom*.9-ominy)/(omaxy-ominy));
        LineTo(hdc,.95*rect.right,(rect.bottom*.9-ominy)/(omaxy-ominy));
        //MoveTo( hdc,.5*rect.right,(rect.bottom*.05-ominy)/(omaxy-ominy));
        //LineTo(hdc,.5*rect.right,(rect.bottom*.95-ominy)/(omaxy-ominy));
        for (i=0;i<=1;i++)
        {
        MoveTo( hdc,(-1*i-ominx)/(omaxx-ominx)*rect.right,(rect.bottom*.88-
        ominy)/(omaxy-ominy));\
        LineTo(hdc,(-1*i-ominx)/(omaxx-ominx)*rect.right,(rect.bottom*.92-
        ominy)/(omaxy-ominy));
        MoveTo( hdc,(1*i-ominx)/(omaxx-ominx)*rect.right,(rect.bottom*.88-
        ominy)/(omaxy-ominy));
        LineTo(hdc,(1*i-ominx)/(omaxx-ominx)*rect.right,(rect.bottom*.92-
        ominy)/(omaxy-ominy));
        }
        EndPaint (hwnd, &ps);
        return 0;
case WM_TIMER :
        hdc = GetDC(hwnd);
        SelectObject (hdc,GetStockObject (WHITE_PEN));
        for ( i=0;i<out_record[1].num_rules_fired;i++)
        {
        MoveTo( hdc,pt[i][0].x,pt[i][0].y);
        LineTo(hdc,pt[i][1].x,pt[i][1].y);
        }
        out_record[1].num_rules_fired = out_record[0].num_rules_fired;
        MoveTo( hdc,AR1.x,AR1.y);
        LineTo(hdc,AR2.x,AR2.y);
        MoveTo( hdc,AR3.x,AR3.y);
        LineTo(hdc,AR2.x,AR2.y);
        LineTo(hdc,AR4.x,AR4.y);
        fuzzy_cs = em->fuzzy.output(et);
        TextOut(hdc,50,10,szbuffer,sprintf(szbuffer,"CS =%.3f    ",em-
        >gai7.output(et)));
        TextOut(hdc,50,30,szbuffer,sprintf(szbuffer,"Fired =%d
        ",out_record[0].num_rules_fired));
```

```
                    SelectObject (hdc,GetStockObject (BLACK_PEN));
                    for ( i=0;i<out_record[0].num_rules_fired;i++)
                    {
                    //TextOut(hdc,10,50+20*i,szbuffer,sprintf(szbuffer,"weight =%f weight*gain
                    =%f",out_record[i].weight_x,out_record[i].weight_y));
                    pt[i][0].x = (out_record[i].weight_x-ominx)/(omaxx-ominx)*rect.right;
                    pt[i][0].y = (rect.bottom*.9-ominy)/(omaxy-ominy)-1;
                    pt[i][1].x = pt[i][0].x;
                    pt[i][1].y = ((.9-fabs(out_record[i].weight_y)*.8)-ominy)/(omaxy-
                    ominy)*rect.bottom-1;
                    //pt[i][1].y = (.9-1*.8)*rect.bottom;
                    MoveTo( hdc,pt[i][0].x,pt[i][0].y);
                    LineTo(hdc,pt[i][1].x,pt[i][1].y);
                    }
                    AR1.x = (fuzzy_cs-ominx)/(omaxx-ominx)*rect.right*.97;
                    AR1.y = (rect.bottom*.94-ominy)/(omaxy-ominy);
                    AR2.x = (fuzzy_cs-ominx)/(omaxx-ominx)*rect.right;
                    AR2.y = (rect.bottom*.9-ominy)/(omaxy-ominy)+1;
                    AR3.x = (fuzzy_cs-ominx)/(omaxx-ominx)*rect.right*1.03;
                    AR3.y = (rect.bottom*.94-ominy)/(omaxy-ominy);
                    AR4.x = (fuzzy_cs-ominx)/(omaxx-ominx)*rect.right;
                    AR4.y = (rect.bottom*.98-ominy)/(omaxy-ominy);
                    MoveTo( hdc,AR1.x,AR1.y);
                    LineTo(hdc,AR2.x,AR2.y);
                    MoveTo( hdc,AR3.x,AR3.y);
                    LineTo(hdc,AR2.x,AR2.y);
                    LineTo(hdc,AR4.x,AR4.y);
                    ReleaseDC (hwnd,hdc);
                    return 0;
            case WM_CLOSE:
                    OutOpen = FALSE;
                    DestroyWindow (hwnd) ;
                    EnableMenuItem(hMainMenu,IDM_OUTRULES,MF_ENABLED);
                    return 0 ;
            }
    return DefWindowProc (hwnd, message, wParam, lParam) ;
    }

long FAR PASCAL _export FuzFilt1MemProc (HWND hwnd, UINT message, UINT wParam,
                                LONG lParam)
    {
    int                 i,j,k=0;
    HDC                 hdc ;
    PAINTSTRUCT         ps ;
    static short        cxChar,cyChar;
    TEXTMETRIC          tm;
    char                char_str[10];
    static RECT         rect;
    HRGN                hrgn;
    DWORD               dword;
    float               Inp_Err;
    static POINT        AR1,AR2,AR3,AR4;
```

```
static HWND             hwndSmaller,hwndLarger;
LPDRAWITEMSTRUCT        lpdis;
char                    szbuffer[50];
switch (message)
        {
        case WM_COMMAND :
                hDerMenu = GetMenu (hwnd);
                GetClientRect(hwnd,&rect);
                switch (wParam)
                {
                case IDC_SMALLER :
                        hdc = GetDC(hwnd);
                        ROA_GAIN -= .0001;
                        em->gai9.set_gain(ROA_GAIN);
                        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",ROA_GAIN));
                        ReleaseDC(hwnd,hdc);
                        return 0;
                case IDC_LARGER :
                        hdc = GetDC(hwnd);
                        ROA_GAIN += .0001;
                        em->gai9.set_gain(ROA_GAIN);
                        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",ROA_GAIN));
                        ReleaseDC(hwnd,hdc);
                        return 0;
                case IDM_START:
                        DerOpen = TRUE;
                        EnableMenuItem(hDerMenu,IDM_START,MF_GRAYED);
                        EnableMenuItem(hDerMenu,IDM_STOP,MF_ENABLED);
                        return 0;
                case IDM_STOP:
                        DerOpen = FALSE;
                        EnableMenuItem(hDerMenu,IDM_STOP,MF_GRAYED);
                        EnableMenuItem(hDerMenu,IDM_START,MF_ENABLED);
                        return 0;
                case IDM_CLOSE:
                        SendMessage(hwnd,WM_CLOSE,0,0);
                        return 0;
                case IDM_GAIN:
                        return 0;
                default:
                        return 0;
                }
        case WM_DRAWITEM :
                lpdis= (LPDRAWITEMSTRUCT) lParam;
                FillRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(LTGRAY_BRUSH));
                FrameRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(BLACK_BRUSH));
                switch (lpdis->CtlID)
                {
                case IDC_SMALLER:
                        Gain_Button(lpdis->hDC,UP);
                        break;
                case IDC_LARGER :
```

```
                Gain_Button(lpdis->hDC,DOWN);
                break;
        }
        return 0;
case WM_SIZE :
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        return 0;
case WM_CREATE :
        hdc= GetDC(hwnd);
        hwndSmaller = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW, 0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_SMALLER,hInst,NULL);
        hwndLarger = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_LARGER,hInst,NULL);
        GetTextMetrics ( hdc, &tm);
        cxChar = tm.tmAveCharWidth;
        cyChar = tm.tmHeight + tm.tmExternalLeading;
        ReleaseDC(hwnd,hdc);
        return 0;
case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;
        GetClientRect(hwnd, &rect);
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",ROA_GAIN));
        SelectObject (hdc,hPen1);
        for (j=0;j<fzz.no_of_inp_regions;j++)
        {
        if ((j == 0) )
        {
        MoveTo( hdc,.1*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,(fzz.inp_mem_fns[1][j].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,(fzz.inp_mem_fns[1][j].b-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        }
        else if (j == fzz.no_of_inp_regions-1)
        {
        MoveTo( hdc,(fzz.inp_mem_fns[1][j].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,(fzz.inp_mem_fns[1][j].b-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        LineTo(hdc,.9*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
        }
         } /* end j */
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        MoveTo( hdc,.05*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,.95*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
```

```
                MoveTo( hdc,.5*rect.right,(rect.bottom*.05-miny)/(maxy-miny));
                LineTo(hdc,.5*rect.right,(rect.bottom*.95-miny)/(maxy-miny));
                for (j=0;j<=4;j++)
                {
                MoveTo( hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.88-
                miny)/(maxy-miny));
                LineTo(hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
                miny)/(maxy-miny));
                MoveTo( hdc,(.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.88-
                miny)/(maxy-miny));
                LineTo(hdc,(.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
                miny)/(maxy-miny));
                }
                EndPaint (hwnd, &ps);
                return 0;
        case WM_TIMER :
                hdc = GetDC(hwnd);
                TextOut(hdc,10,90,szbuffer,sprintf(szbuffer,"adv cs=%.3f ",-em-
                >gai10.output(et)));
                TextOut(hdc,10,110,szbuffer,sprintf(szbuffer,"roa=%.3f ",ROA));
                TextOut(hdc,10,130,szbuffer,sprintf(szbuffer,"e1=%.3f ",e1));
                SelectObject (hdc,GetStockObject (WHITE_PEN));
                MoveTo( hdc,AR1.x,AR1.y);
                LineTo(hdc,AR2.x,AR2.y);
                MoveTo( hdc,AR3.x,AR3.y);
                LineTo(hdc,AR2.x,AR2.y);
                LineTo(hdc,AR4.x,AR4.y);
                ROA = em->gai9.output(et);
                SelectObject (hdc,GetStockObject (BLACK_PEN));
                AR1.x = (ROA-minx)/(maxx-minx)*rect.right*.97;
                AR1.y = (rect.bottom*.94-miny)/(maxy-miny);
                AR2.x = (ROA-minx)/(maxx-minx)*rect.right;
                AR2.y = (rect.bottom*.9-miny)/(maxy-miny)+1;
                AR3.x = (ROA-minx)/(maxx-minx)*rect.right*1.03;
                AR3.y = (rect.bottom*.94-miny)/(maxy-miny);
                AR4.x = (ROA-minx)/(maxx-minx)*rect.right;
                AR4.y = (rect.bottom*.98-miny)/(maxy-miny);
                MoveTo( hdc,AR1.x,AR1.y);
                LineTo(hdc,AR2.x,AR2.y);
                MoveTo( hdc,AR3.x,AR3.y);
                LineTo(hdc,AR2.x,AR2.y);
                LineTo(hdc,AR4.x,AR4.y);
                ReleaseDC (hwnd,hdc);
                return 0;
        case WM_CLOSE:
                DerOpen = FALSE;
                DestroyWindow (hwnd) ;
                EnableMenuItem(hMainMenu,IDM_FILTER,MF_ENABLED);
                return 0 ;
                }
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

```
long FAR PASCAL _export FuzFilt2MemProc (HWND hwnd, UINT message, UINT wParam,
                                LONG lParam)
        {
        int                     i,j,k=0;
        HDC                     hdc ;
        PAINTSTRUCT             ps ;
        static short            cxChar,cyChar;
        TEXTMETRIC              tm;
        char                    char_str[10];
        static RECT             rect;
        HRGN                    hrgn;
        DWORD                   dword;
        float                   Inp_Err;
        static POINT            AR1,AR2,AR3,AR4;
        static HWND             hwndSmaller,hwndLarger,hwndOutSmaller,hwndOutLarger;
        LPDRAWITEMSTRUCT        lpdis;
        char                    szbuffer[50];

        switch (message)
                {
                case WM_COMMAND :
                        hDerMenu = GetMenu (hwnd),
                        GetClientRect(hwnd,&rect);
                        switch (wParam)
                        {
                        GetClientRect(hwnd,&rect);
                        case IDC_SMALLER :
                                hdc = GetDC(hwnd);
                                ERR_GAIN -= .001;
                                em->gai8.set_gain(ERR_GAIN);
                                TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",ERR_GAIN));
                                ReleaseDC(hwnd,hdc);
                                return 0;
                        case IDC_LARGER :
                                hdc = GetDC(hwnd);
                                ERR_GAIN += .001;
                                em->gai8.set_gain(ERR_GAIN);
                                TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",ERR_GAIN));
                                ReleaseDC(hwnd,hdc);
                                return 0;
                        case IDC_OUTSMALLER :
                                hdc = GetDC(hwnd);
                                FUZGAIN -= 1.0;
                                em->gai10.set_gain(FUZGAIN);
                                TextOut(hdc,rect.right-50,70,szbuffer,sprintf(szbuffer,"%.1f
                                ",FUZGAIN));
                                ReleaseDC(hwnd,hdc);
                                return 0;
                        case IDC_OUTLARGER :
                                hdc = GetDC(hwnd);
                                FUZGAIN += 1.0;
```

```
                em->gai10.set_gain(FUZGAIN);
                TextOut(hdc,rect.right-50,70,szbuffer,sprintf(szbuffer,"%.1f
                ",FUZGAIN));
                ReleaseDC(hwnd,hdc);
                return 0;
        case IDM_START:
                DerOpen = TRUE;
                EnableMenuItem(hDerMenu,IDM_START,MF_GRAYED);
                EnableMenuItem(hDerMenu,IDM_STOP,MF_ENABLED);
                return 0;
        case IDM_STOP:
                DerOpen = FALSE;
                EnableMenuItem(hDerMenu,IDM_STOP,MF_GRAYED);
                EnableMenuItem(hDerMenu,IDM_START,MF_ENABLED);
                return 0;
        case IDM_CLOSE:
                SendMessage(hwnd,WM_CLOSE,0,0);
                return 0;
        case IDM_GAIN:
                return 0;
        default:
                return 0;
        }
case WM_DRAWITEM :
        lpdis= (LPDRAWITEMSTRUCT) lParam;
        FillRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(LTGRAY_BRUSH));
        FrameRect (lpdis->hDC,&lpdis->rcItem,GetStockObject(BLACK_BRUSH));
        switch (lpdis->CtlID)
        {
        case IDC_SMALLER:
                Gain_Button(lpdis->hDC,UP);
                break;
        case IDC_LARGER :
                Gain_Button(lpdis->hDC,DOWN);
                break;
        case IDC_OUTSMALLER:
                Gain_Button(lpdis->hDC,UP);
                break;
        case IDC_OUTLARGER :
                Gain_Button(lpdis->hDC,DOWN);
                break;
        }
        return 0;
case WM_SIZE :
        GetClientRect(hwnd,&rect);
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndOutSmaller,rect.right-
        50,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndOutLarger,rect.right-
        50,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
```

```
        return 0;
case WM_CREATE :
        hdc= GetDC(hwnd);
        hwndSmaller = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW, 0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_SMALLER,hInst,NULL);
        hwndLarger = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_LARGER,hInst,NULL);
        hwndOutSmaller = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW, 0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_OUTSMALLER,hInst,NULL);
        hwndOutLarger = CreateWindow("button","",
         WS_CHILD | WS_VISIBLE | BS_OWNERDRAW,0,0,
         BTN_WIDTH,BTN_HEIGHT,hwnd,IDC_OUTLARGER,hInst,NULL);
        GetTextMetrics ( hdc, &tm);
        cxChar = tm.tmAveCharWidth;
        cyChar = tm.tmHeight + tm.tmExternalLeading;
        ReleaseDC(hwnd,hdc);
        return 0;
case WM_PAINT :
        hdc = BeginPaint (hwnd, &ps) ;
        GetClientRect(hwnd, &rect);
        MoveWindow
        (hwndSmaller,10,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndLarger,10,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndOutSmaller,rect.right-
        50,20+BTN_HEIGHT,BTN_WIDTH,BTN_HEIGHT,TRUE);
        MoveWindow (hwndOutLarger,rect.right-
        50,10,BTN_WIDTH,BTN_HEIGHT,TRUE);
        TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",ERR_GAIN));
        TextOut(hdc,rect.right-50,70,szbuffer,sprintf(szbuffer,"%.1f ",FUZGAIN));
        SelectObject (hdc,GetStockObject (BLACK_PEN));
        MoveTo( hdc,.05*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        LineTo(hdc,.95*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
        MoveTo( hdc,.5*rect.right,(rect.bottom*.05-miny)/(maxy-miny));
        LineTo(hdc,.5*rect.right,(rect.bottom*.95-miny)/(maxy-miny));
        for (j=0;j<=4;j++)
        {
        MoveTo( hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.88-
        miny)/(maxy-miny));
        LineTo(hdc,(-.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
        miny)/(maxy-miny));
        MoveTo( hdc,(.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.88-
        miny)/(maxy-miny));
        LineTo(hdc,(.25*j-minx)/(maxx-minx)*rect.right,(rect.bottom*.92-
        miny)/(maxy-miny));
        }
        SelectObject (hdc,hPen1);
        //display memberships
        MoveTo(hdc,(fzz.inp_mem_fns[0][0].a-minx)/(maxx-
        minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
```

```
                LineTo(hdc,(fzz.inp_mem_fns[0][0].c-minx)/(maxx-
                minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
                LineTo(hdc,(fzz.inp_mem_fns[0][0].d-minx)/(maxx-
                minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
                MoveTo( hdc,(fzz.inp_mem_fns[0][1].a-minx)/(maxx-
                minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
                LineTo(hdc,(fzz.inp_mem_fns[0][1].c-minx)/(maxx-
                minx)*rect.right,(rect.bottom*.1-miny)/(maxy-miny));
                LineTo(hdc,(fzz.inp_mem_fns[0][1].d-minx)/(maxx-
                minx)*rect.right,(rect.bottom*.9-miny)/(maxy-miny));
                EndPaint (hwnd, &ps);
                return 0;
        case WM_TIMER :
                hdc = GetDC(hwnd);
                SelectObject (hdc,GetStockObject (WHITE_PEN));
                MoveTo( hdc,AR1.x,AR1.y);
                LineTo(hdc,AR2.x,AR2.y);
                MoveTo( hdc,AR3.x,AR3.y);
                LineTo(hdc,AR2.x,AR2.y);
                LineTo(hdc,AR4.x,AR4.y);
                e1 = em->gai8.output(et);
                SelectObject (hdc,GetStockObject (BLACK_PEN));
                AR1.x = (e1-minx)/(maxx-minx)*rect.right*.97;
                AR1.y = (rect.bottom*.94-miny)/(maxy-miny);
                AR2.x = (e1-minx)/(maxx-minx)*rect.right;
                AR2.y = (rect.bottom*.9-miny)/(maxy-miny)+1;
                AR3.x = (e1-minx)/(maxx-minx)*rect.right*1.03;
                AR3.y = (rect.bottom*.94-miny)/(maxy-miny);
                AR4.x = (e1-minx)/(maxx-minx)*rect.right;
                AR4.y = (rect.bottom*.98-miny)/(maxy-miny);
                MoveTo( hdc,AR1.x.AR1.y);
                LineTo(hdc,AR2.x,AR2.y);
                MoveTo( hdc,AR3.x,AR3.y);
                LineTo(hdc,AR2.x,AR2.y);
                LineTo(hdc,AR4.x,AR4.y);
                TextOut(hdc,rect.right-50,70,szbuffer,sprintf(szbuffer,"%.1f ",FUZGAIN));
                TextOut(hdc,10,70,szbuffer,sprintf(szbuffer,"%.4f ",ERR_GAIN));
                ReleaseDC (hwnd,hdc);
                return 0;
        case WM_CLOSE:
                DerOpen = FALSE;
                DestroyWindow (hwnd) ;
                EnableMenuItem(hMainMenu,IDM_FILTER,MF_ENABLED);
                return 0 ;
        }
return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

```
//FILENAME: Timer.CPP
//BY:     Craig Jensen

#include "timer.h"

timer::timer ( void )
 {
 outportb (Base_Addr + 0x0A ,0x02 );  // connect 100KHz clock to counter 0
 outportb (Base_Addr + 3 ,0x00 );  // set OP0 low
 outportb (Base_Addr + 0x0F ,0x30); //initialize counter/timer
 outportb (Base_Addr + 0x0C ,0xFF); // init counter LSB
 outportb (Base_Addr + 0x0C ,0xFF); // init counter MSB
 outportb (Base_Addr + 3 ,0x01 );  // gate counter 0 (write out0 high) (starts counter 0)
}

timer::~timer ( void )
 {
 outportb (Base_Addr + 3 ,0x00 ); //remove gate
 }

word
timer::get_count ( void )
   {
   byte    low_byte, high_byte;
   word    result;

   outportb (Base_Addr + 0x0F , 0x00);  // latch counter 0

   low_byte  = inportb (Base_Addr + 0x0C);  // read LSB
   high_byte = inportb (Base_Addr + 0x0C);  // read MSB

   result = (high_byte * 256) + low_byte ;
   return result;
   }

float
timer::get_elapsed_time( void)
   {
   float elapsed_time;
   curr_count = get_count();
   if (curr_count > last_count)     // counter overran
     elapsed_time = float ((65535-curr_count) + last_count) * 0.00001;
   if (curr_count <= last_count)
     elapsed_time = float ((last_count - curr_count)) * 0.00001; // 100KHz clock
     last_count = curr_count;
   return elapsed_time;
   }
```

```
//FILENAME:  Timer.H
//BY:        Craig Jensen

# ifndef  Timer_INCLUDED
# define  Timer_INCLUDED

typedef unsigned char byte;
typedef unsigned int  word;

const word Base_Addr = 0x0300;

# include <dos.h>

class timer
  {
   protected :

            byte high_byte;
            byte low_byte;
   word curr_count,last_count;


   public :

                        timer ( void );
                        ~timer ( void );
                        word  get_count ( void );
      float get_elapsed_time(void);

  };

# endif
```

```
/*  File UT.H  Header file for utility UT*.C files *

#ifndef UT_H
#define UT_H

//#include <stdio.h>
#include "uttypes.h"

/* UTMATRIX.C */
float **matrix2d (int rows_lower_lim,int rows_upper_lim,
  int cols_lower_lim,int cols_upper_lim);
float ***matrix3d (int rows_lower_lim,int rows_upper_lim,
  int cols_lower_lim,int cols_upper_lim,int n_lower_lim,int n_upper_lim);
float ****matrix4d (int rows_lower_lim,int rows_upper_lim,
  int cols_lower_lim,int cols_upper_lim,
  int m_lower_lim,int m_upper_lim,
  int n_lower_lim,int n_upper_lim);
void rand_init_3d_matrix (float ***w,int l_rows,int l_cols,int n_inp,
   float range);
void init_3d_matrix (float ***w,int l_rows,int l_cols,int n_inp,
   const float val);
void rand_init_4d_matrix (float ****w,int l_rows,int l_cols,int m_inp,
  int n_inp, float range);
void rand_init_2d_matrix (float **w, int l_rows, int l_cols,float range);
void init_2d_matrix (float **a, int l_rows, int l_cols, const float val);
void free_matrix2d (float **a,int rows_lower_lim,int cols_lower_lim);
void free_matrix3d (float ***a,int rows_lower_lim,int cols_lower_lim,
  int n_lower_lim);
void free_matrix4d (float ****a,int rows_lower_lim,int cols_lower_lim,
  int m_lower_lim,int n_lower_lim);
short write_2d_matrix (float **w,int l_rows,int l_cols,char *file_name);
void print_2d_matrix (float **w,int l_rows,int l_cols);
short read_2d_matrix (float **w,char *file_name,int rows,int cols);
short write_3d_matrix (float ***w,int l_rows,int l_cols,int n_inp,
  char *file_name);
void print_3d_matrix (float ***w,int rows,int cols,int inp);
void print_4d_matrix (float ****w,int rows,int cols,int inp,int inp2);
short read_3d_matrix (float ***w,char *file_name,int rows,int cols,int inp);
short write_4d_matrix (float ****w,int l_rows,int l_cols,int m_inp,
  int l_inp,char *file_name);
short read_4d_matrix (float ****w,char *file_name,int rows,int cols,
  int inp,int inp2);

/* UTFILES.C */
long open_input_text_file (FILE **the_file,path_str file_name);
short open_output_text_file (FILE **the_file,path_str file_name);
int fcopy (const char far *,const char far*);

/*  UTPRCS.C  */
char get_answer (char chr1,char chr2,char chr3,char chr4);
int cget_int (int len,int *value);
int cget_float (int len,float *value);
```

```
void  shuffl (int no_of_items,int *pshuffle_list);
float frand (void);

/* UTVECT.C */
ui_vector allocate_ui_vector (int low_limit,int up_limit);
f_vector allocate_f_vector (int low_limit,int up_limit);
d_vector allocate_d_vector (int low_limit,int up_limit);
void copy_ui_vector (ui_vector source,ui_vector dest,int start,int len);
void copy_f_vector (f_vector source,f_vector dest,int start,int len);
void copy_d_vector (d_vector source,d_vector dest,int start,int len);
void free_ui_vector (ui_vector v,int low_limit);
void free_f_vector (f_vector v,int low_limit);
void free_d_vector (d_vector v,int low_limit);

#endif
```

```
/* File UTFILES.CPP  File handling procedures: Open text files, copy file  */

#ifndef UTFILES_C
#define UTFILES_C

#include <stdio.h>
#include <conio.h>
#include <dir.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>
#include <windows.h>

#include "ut.h"

long open_input_text_file (FILE **the_file,path_str file_name);
short open_output_text_file (FILE **the_file,path_str file_name);
int fcopy (const char *source_name,const char *dest_name);

/*  Implementation  */

void file_error_message (path_str file_name,char *in_out) {
        char er[100];
        sprintf (er,"Problem opening %s file %s.\0",in_out,file_name);
        MessageBox(NULL,er,"Error",MB_OK);
        //clrscr ();
        //gotoxy (1,9);
        //cprintf ("Problem opening %s file %s.\r\n",in_out,file_name);
        //cprintf ("Hit key to continue: ");
        //getch();

        return;
        }

long open_input_text_file (FILE **the_file,path_str file_name) {
        /* Returns file size in bytes if successful, 0 otherwise */
        struct ffblk file_rec;
        char buffer[100];
        if ((!file_name) || (file_name[0] == '\0')) {
                sprintf (buffer,"Input file name is blank.\0");
                MessageBox(NULL,buffer,"Error",MB_OK);
                //getch();
                return 0;
                }
        if (findfirst(file_name,&file_rec,0) == 0) {
                *the_file = fopen(file_name,"rt");
                return file_rec.ff_fsize;
                }
        else file_error_message (file_name,"input");
        return 0;
        } /* end func */
```

```
short open_output_text_file (FILE **the_file,path_str file_name) {
        /* Returns 1 if successful open */
        if ((!file_name) || (file_name[0] == '\0')) {
                //cprintf ("\r\n\nOutput file name is blank.");
                MessageBox(NULL,"Output file name is blank","Save",MB_OK);
                return 0;
                }
        if ((*the_file = fopen(file_name,"wt")) == NULL) {
                file_error_message (file_name,"output");
                return 0;
                }
         return 1; /* Successful */
         } /* end func */

int fcopy (const char *source_name,const char *dest_name) {
   /* General file copying utility */
   /* Returns 0 on successful copy */
   int source_file,dest_file;
   long int file_size,i,no_of_blocks,remainder;
   ldiv_t result;
   char data[200],one_byte;
   long int block_size = sizeof(data);
   if ((source_file = open (source_name,O_RDONLY | O_BINARY)) == -1)
     return -1;
   if ((dest_file = open (dest_name,O_WRONLY | O_CREAT | O_TRUNC | O_BINARY,
     S_IREAD | S_IWRITE)) == -1) return -2;
   file_size = filelength (source_file);
   result = ldiv (file_size,block_size);
   no_of_blocks = result.quot;
   remainder = result.rem;
   for (i=1;i<=no_of_blocks;i++) {
         read (source_file,data,block_size);
         write (dest_file,data,block_size);
         }
   for (i=0;i<remainder;i++) {
         read (source_file,&one_byte,1);
         write (dest_file,&one_byte,1);
         }
   close (source_file);
   close (dest_file);
   return 0;
   }

#endif
```

REFERENCES

[1] Uwe Kiencke, "A View of Automotive Control Systems," IEEE Control Systems Magazine, August, 1988.

[2] Davorin Hrovat, "Computer Control Systems for Automotive Power Trains," IEEE Control Systems Magazine, August, 1988.

[3] Junichi Ishii, et. al., "Wide Range Air-Fuel Ratio Control System," SAE Paper 880134.

[4] Angelo S. Martinez, et.al., "Design of Fuzzy Logic based Engine Idle-Speed Controllers,"

[5] George Vachtsevanos, et. al., "Fuzzy Logic Control of an Automotive Engine," IEEE Control Systems Magazine, June 1993.

[6] B.J. Lee, et. al. "Engine Control Using Aticipitory Band in the Sliding Phase Plane," IEEE Transactions on Control Systems Technology, Vol. 1, No. 4, December 1993.

[7] Jeffrey A. Cook, et. al., "Modeling of an Internal Combustion Engine for Control Analysis" IEEE Control Systems Magazine, August 1988.

[8] John F. Cassidy, et. al., "On the design of Elecrtonic Automotive Engine Controls Using Linear Quadratic Control Theory" IEEE Trans. Auto. Contr., vol. AC-25, no. 5, Oct. 1980.

[9] R. L. Morris, et. al., "An Identification Approach to Throttle-Torque Modeling" SAE Paper 810448, 1981.

[10] Katsuhiko Ogata, Modern Control Engineering 2nd Edition, Englewood Cliffs, NJ: Prentice Hall, 1990.

[11] Charles L. Phillips, et. al., Digital Control System, Analysis and Design, Englewood Cliffs, NJ: Prentice Hall, 1990.

[12] Omron Electronics, Inc., "An introduction to Fuzzy Logic and its Applications in Control Systems," 1991.

[13] Glenn Anderson, Omron Electronics, Inc., "Fundamentals of Fuzzy Logic," Sensors, March 1993.

[14] Stephen T. Welstead, Neural Network and Fuzzy Logic Applications in C/C++, New York, NY: John Wiley & Sons, Inc., 1994.

[15] Earl Cox, "Adaptive Fuzzy Systems," IEEE Spectrum, February 1993.

[16] M. Abate, et. al., "Application of Some New Tools to Robust Stability Analysis of Spark Ignition Engines: A Case Study," IEEE Trans. on Contr. Sys. Technology, Vol. 2, No. 1, March 1994.

[17] Hisashi Matsumoto, et. al., "Application of Fuzzy Control to Internal Combustion Engines," JMSE Internaltional Journal, Series B, Vol. 37, No. 1, 1994.

[18] Paul S. Min, "Diagnosis of On-Board Sensors in Internal Combustion Engines," 38th IEEE Vehicle Technology Conference, June 15-17, Philadelphia, PA.

[19] Manu Malek-Zavarei, et. al., TIME-DELAY SYSTEMS Analysis, Optimization and Applications, North Holland, 1987.

[20] P. S. Min, "Detection of Incipient Failures in Dynamic Systems," Ph. D. Thesis, the University of Michigan, 1987.

[21] Jack J. McCauley, "A Fuzzy-Logic Torque Servo," The C Users Journal, March 1994.

[22] David I. Brubaker, "Fuzzy-logic system solves control problem," Sensors, May 1993.

[23] Naoki Tomisawa, et. al., "Trends in Electronic Engine Control and Development of Optimal Microcomputers," SAE Paper 880136.

[24] Twente University of Technology SIMulation (TUTSIM) program, Applied I, Palo Alto, California

[25] Dobner, D.J. (1983) "Dynamic engine models for control development-Part I: Non-Linear and linear model formulation," Int. J. of Vehicle Design, Technological Adavces in Vehicle design Series, SP4, Application of Control Theory in the Automotive Industry.